Your feedback is valuable, and I've integrated the daily breakdown from your prompt into my previous, more detailed implementation cycle. This new, combined format should be much easier for an AI agent to follow.

Here is the updated implementation, now including **daily breakdowns** for a step-by-step approach.

**Phase 1: Foundation & Secure Authentication (Weeks 1-4)**

**Week 1: Basic App Infrastructure**

**Day 1-2: Project Setup**

* **Task:** Project Initialization & Dependencies
  + **Description:** Create the core Flutter project and add essential libraries.
  + **Implementation:** Run flutter create smartsheba. In pubspec.yaml, add flutter\_bloc, go\_router, shared\_preferences, and http. This provides the foundation for state management, navigation, network requests, and local caching.
  + **Synchronization Notes:** This is the absolute starting point. All future work will be done within this new project directory.
* **Task:** Folder Structure Creation
  + **Description:** Organize the project with a feature-first, layered architecture.
  + **Implementation:** Create the directories lib/core/ for shared utilities and lib/features/ for self-contained features like auth and home. This ensures a scalable and maintainable codebase.

**Day 3-4: Theming & Navigation**

* **Task:** App Theme Configuration
  + **Description:** Define a robust theme system for UI consistency.
  + **Implementation:** Create lib/core/theme/colors.dart and lib/core/theme/app\_theme.dart to define our color palette and theme data. This theme will be used by every UI component.
* **Task:** Basic Navigation Setup
  + **Description:** Set up the application's routing using go\_router.
  + **Implementation:** In lib/routes.dart, define the initial routes (/, /login) and implement a **conditional redirect** to handle navigation based on the user's authentication status. This ensures a secure navigation flow from the start.

**Day 5-7: Authentication Foundation**

* **Task:** Core Data Models & BLoC Setup
  + **Description:** Create the core data model and state management for authentication.
  + **Implementation:** Create lib/features/auth/domain/entities/user\_entity.dart. The UserEntity will have a role field (enum Role { customer, provider, admin }), which is the heart of our **RBAC system**. Implement AuthBloc to manage authentication states (Authenticated(User user), Unauthenticated).
  + **Synchronization Notes:** This is the most critical step for RBAC. The role field will be the single source of truth for user permissions.
* **Task:** Login/Register UI Screens
  + **Description:** Design the user interfaces for authentication.
  + **Implementation:** Create the UI for the login, register, and OTP verification screens, focusing on a clean and intuitive user experience.

**Week 2: User Authentication System**

**Day 8-10: Authentication Logic**

* **Task:** Phone Number Validation & OTP Integration
  + **Description:** Implement the logic to validate phone numbers and handle OTP.
  + **Implementation:** Add phone number validation logic. For development, use a dummy OTP service. On the backend, upon successful OTP verification, an API must return the user's token and their **role**.
  + **Synchronization Notes:** This step populates the role field from the backend, making it available for use throughout the application.
* **Task:** Local Storage Setup
  + **Description:** Securely save the authentication token and user role on the device.
  + **Implementation:** Use shared\_preferences to save the authentication token. This allows the user to stay logged in and enables the app to retrieve their role on startup.

**Day 11-14: User Profile**

* **Task:** Profile Creation & Management
  + **Description:** Create screens for new users to set up their profiles and for existing users to manage them.
  + **Implementation:** Build the profile creation screen where new users, who are automatically assigned the **customer role**, can input their details. Implement the edit profile functionality.
  + **Synchronization Notes:** These screens update the UserEntity created in Week 1, ensuring data consistency.

**Week 3: Service Catalog Foundation**

**Day 15-17: Service Categories**

* **Task:** Service Category Model & Home Screen Layout
  + **Description:** Define the data model for service categories and design the home screen.
  + **Implementation:** Create a ServiceCategory model. Design the home screen to dynamically display service categories in a grid, with a search bar and a location display.
  + **Synchronization Notes:** This builds on the UI foundation from Week 1 and is the first interactive content layer for the app.

**Day 18-21: Basic Service Discovery**

* **Task:** Service List & Detail Screens
  + **Description:** Create pages to display a list of services and their details.
  + **Implementation:** Build a ServiceListScreen to show services within a category and a ServiceDetailScreen for more information. **Crucially, the "Contact provider" button on the service screen is wrapped in a conditional widget that only renders for authenticated users with the customer role.**
  + **Synchronization Notes:** This is a key RBAC implementation, ensuring only authorized users can initiate the booking process.

**Week 4: Provider Management**

**Day 22-24: Provider Profiles**

* **Task:** Provider Model & List Display
  + **Description:** Define the provider data model and how their profiles are displayed.
  + **Implementation:** Create a ServiceProvider model. Implement provider list and detail screens, which display provider cards with their rating and basic information.
* **Task:** Provider Dashboard Structure
  + **Description:** Design the dashboard for providers.
  + **Implementation:** Create a ProviderDashboard screen that is **only accessible to users with the provider role**, as enforced by the navigation logic from Week 2.

**Day 25-28: Provider Registration**

* **Task:** Provider Registration Form & Role-Based Navigation
  + **Description:** Implement the application process for users to become providers.
  + **Implementation:** Create a multi-step form for providers to submit their information and verification documents. This application is submitted as a **request for a role change**. The user's role remains customer until an admin (handled in Week 16) manually approves them.
  + **Synchronization Notes:** This is the culmination of the RBAC system. It defines the secure process for a user to transition to a new role, which is later managed in the Admin Panel.

.

Of course. Here is the next part of the implementation cycle, picking up where we left off. This section details **Phase 2: Core Booking System**, which is the heart of the SmartSheba application. Every step is fully descriptive and includes proper synchronization with the foundational work from Phase 1.

**Phase 2: Core Booking System (Weeks 5-8)**

This phase focuses on building the core transaction engine of the app, allowing customers to book services and providers to manage those bookings. It heavily relies on the user roles and data models established in Phase 1.

**Week 5: Basic Booking Creation**

**Day 29-31: Booking Foundation**

* **Task:** Booking Model & Backend API
  + **Description:** Define the Booking data model and create the backend API endpoint for creating a new booking.
  + **Implementation:** Create lib/features/booking/domain/entities/booking\_entity.dart. The model must include fields like id, customerId, providerId, serviceCategory, scheduledAt, and status. The backend endpoint /bookings will receive a POST request with the booking details and return a confirmation.
  + **Synchronization Notes:** The customerId field will be populated from the UserEntity of the authenticated customer, ensuring the booking is correctly attributed.
* **Task:** Customer-Facing Booking UI
  + **Description:** Build the UI screens for the customer to select a service and schedule a booking.
  + **Implementation:** Create lib/features/booking/presentation/pages/book\_service\_page.dart. This page is accessed from the ServiceDetailScreen (Week 3) and includes a date/time picker, a form for customer details, and a review/confirmation screen. The "Book" button is **only visible to authenticated customer users**, as per the RBAC check from Week 3.
  + **Synchronization Notes:** This UI is directly linked to the service details and utilizes the UserEntity to p re-fill customer information, ensuring a smooth UX.

**Day 32-35: Booking Management**

* **Task:** My Bookings Screen
  + **Description:** Create a screen for customers to view all their past and upcoming bookings.
  + **Implementation:** Build lib/features/booking/presentation/pages/my\_bookings\_page.dart. This screen will fetch a list of bookings associated with the current user's id. Implement sorting by status (e.g., upcoming, completed) and display status indicators.
  + **Synchronization Notes:** This screen's data is retrieved from the backend using the id from the AuthBloc's Authenticated state, ensuring proper data isolation.
* **Task:** Booking Status Updates
  + **Description:** Implement the logic to change booking statuses.
  + **Implementation:** Create a backend endpoint /bookings/{id}/status that only accepts requests from the provider role. This endpoint updates the booking status in the database.
  + **Synchronization Notes:** This is a critical RBAC enforcement point. The backend must verify that the user making the request has the correct role before allowing the status change, preventing a customer from marking a service as completed.

**Week 6: Provider Booking Management**

**Day 36-38: Provider Booking Views**

* **Task:** Incoming Requests Screen
  + **Description:** Build the screen for providers to view new booking requests.
  + **Implementation:** Create lib/features/booking/presentation/pages/incoming\_requests\_page.dart. This screen is **only accessible to the provider role** via the navigation setup from Week 2. It fetches all pending bookings associated with the authenticated provider's ID.
  + **Synchronization Notes:** The entire UI is gated by the RBAC system. The API endpoint for fetching provider-specific requests uses the authenticated providerId.
* **Task:** Accept/Decline Logic
  + **Description:** Implement the functionality for providers to accept or decline a booking request.
  + **Implementation:** Add "Accept" and "Decline" buttons to the incoming request UI. The corresponding API calls to update the booking status are **protected on the backend to accept only requests from the provider role**.
  + **Synchronization Notes:** This directly uses the booking status update logic from Week 5, ensuring a seamless data flow.

**Day 39-42: Basic Communication**

* **Task:** In-App Chat Foundation
  + **Description:** Define the data model for chat messages and build the chat UI.
  + **Implementation:** Create lib/features/chat/domain/entities/chat\_message.dart. The ChatScreen UI should include a message list view and a text input field.
* **Task:** Chat Integration

**Description:** Link the chat to specific bookings.

* + **Implementation:** The chat is initiated from a booking detail screen. The backend must verify that the user accessing the chat API is either the customerId or providerId of that specific booking. This prevents unauthorized access.
  + **Synchronization Notes:** This feature builds on the Booking entity from Week 5 and the user roles from Week 1 to provide a secure and contextual communication channel.

**Week 7: Payment & Booking Completion**

**Day 43-45: Payment Foundation**

* **Task:** Payment Method Selection & Status Tracking
  + **Description:** Implement the UI for selecting a payment method and tracking payment status.
  + **Implementation:** Create lib/features/payment/presentation/pages/payment\_page.dart. Include options for "Cash on Delivery" and placeholder UI for "Mobile Banking" options. The Booking model's status should be updated to PaymentPending upon booking confirmation.
* **Task:** Basic Pricing System
  + **Description:** Implement a simple pricing system.
  + **Implementation:** Define a price field in the Booking entity. Initially, this can be a static value or a basic calculation.

**Day 46-49: Booking Completion**

* **Task:** Service Completion Flow
  + **Description:** Implement the flow for a provider to mark a service as complete.
  + **Implementation:** Add a "Mark as Completed" button to the ProviderDashboard for active bookings. This button is **only visible and functional for the provider role**.
  + **Synchronization Notes:** This action triggers a final booking status change (completed), which is another critical RBAC check.
* **Task:** Rating and Review System
  + **Description:** Allow customers to rate and review providers after a service is completed.
  + **Implementation:** Create a Review data model. After a booking is marked as completed, the customer is prompted to leave a rating. The review submission form is **only available to the customer role**.

**Week 8: Basic Location Services**

**Day 50-52: Location Foundation**

* **Task:** Location Permission & Map Integration
  + **Description:** Handle device location permissions and embed a basic map.
  + **Implementation:** Use a package like geolocator to handle location permissions. Embed a static map to display locations.
* **Task:** Address Input System
  + **Description:** Create a system for users to input their service address.
  + **Implementation:** Build a screen for manual address entry with basic validation and a map to confirm the location.

**Day 53-56: Service Area Management**

* **Task:** Provider Location Setup & Nearby Services
  + **Description:** Allow providers to define their service area and enable customers to find nearby services.
  + **Implementation:** Implement a screen for the **provider role** to define their service area. On the customer's home screen, add a filter to sort or find services based on proximity to their current location.
  + **Synchronization Notes:** This feature combines location data with the ServiceProvider model from Week 4 to provide a more refined service discovery experience.

Here's the next phase of the SmartSheba implementation, focusing on advanced features that enhance the user experience and platform functionality.

**Phase 3: Advanced Features (Weeks 9-12)**

This phase builds on the core booking system by introducing smart search, a robust notification system, and basic AI features. The focus is on making the app more intelligent and user-friendly.

**Week 9: Search and Filtering**

**Day 57-59: Advanced Search**

* **Task Name:** Smart Search Implementation
  + **Description:** Upgrade the basic search functionality to be more intelligent and user-friendly.
  + **Implementation Details:** Implement a search service that queries the backend for services and providers. This search should include auto-complete suggestions and fuzzy matching.
  + **Synchronization Notes:** This uses the ServiceCategory and ServiceProvider models from Week 3 and 4, respectively, to provide a more refined search experience.
  + **Role Impact:** customer, provider (as it improves provider discoverability).
* **Task Name:** Voice Search (Basic)
  + **Description:** Add a voice-to-text input option for the search bar.
  + **Implementation Details:** Integrate a package like speech\_to\_text to enable a voice search button on the home screen. The spoken input is converted to text and used as the search query.
  + **Synchronization Notes:** This UI feature directly connects to the search logic implemented in the previous task.
  + **Role Impact:** customer.

**Day 60-63: Filtering System**

* **Task Name:** Advanced Filters
  + **Description:** Implement a comprehensive filtering system to help users find services more easily.
  + **Implementation Details:** Add filters for price range, rating, distance, and availability. These filters will send multiple parameters to the backend API to narrow down search results.
  + **Synchronization Notes:** These filters work on the data fetched by the API calls, providing a more detailed way to interact with the service and provider data.
  + **Role Impact:** customer.

**Week 10: Notifications System**

**Day 64-66: Push Notifications**

* **Task Name:** Firebase Setup & Push Notifications
  + **Description:** Integrate Firebase Cloud Messaging (FCM) to send real-time notifications to users.
  + **Implementation Details:** Set up a Firebase project and integrate firebase\_messaging into the app. Configure the backend to send targeted notifications to specific device tokens.
  + **Synchronization Notes:** This upgrades the basic notification system from Week 5. The backend will use the user's role to send different notification types—e.g., a "new booking" notification for a provider or a "booking confirmed" notification for a customer.
  + **Role Impact:** All roles.
* **Task Name:** Notification Handling
  + **Description:** Implement logic to handle notifications when the app is in the foreground, background, and terminated states.
  + **Implementation Details:** Implement onMessage, onMessageOpenedApp, and onBackgroundMessage listeners to handle different notification states and navigate the user to the relevant screen upon tap.
  + **Synchronization Notes:** Navigation links from notifications should lead to the correct pages, such as a booking detail screen, a chat screen, or a promotion page.
  + **Role Impact:** All roles.

**Day 67-70: In-App Notifications**

* **Task Name:** Notification Center
  + **Description:** Create a dedicated screen for users to view all their notifications.
  + **Implementation Details:** Build a list view to display all notifications with a "mark as read" feature. Notifications can be categorized for easy management.
  + **Synchronization Notes:** This provides a centralized hub for all notifications, complementing the push notifications.
  + **Role Impact:** All roles.

**Week 11: User Experience Enhancements**

**Day 71-73: Onboarding**

* **Task Name:** App Introduction Screens
  + **Description:** Create an engaging onboarding experience for new users.
  + **Implementation Details:** Design a carousel of screens that highlight the app's main features and benefits. Request necessary permissions (like location) at the appropriate time.
  + **Synchronization Notes:** This is a UI polish that runs once for new users. After completion, it transitions to the login/registration screen, using the authentication logic from Week 2.
  + **Role Impact:** All roles.
* **Task Name:** Tutorial System
  + **Description:** Implement a guide for first-time users.
  + **Implementation Details:** Use interactive tooltips or an overlay to guide users through key actions, such as booking a service or accepting a request. The tutorial should be **role-specific**, guiding a customer on how to book and a provider on how to manage requests.
  + **Synchronization Notes:** This feature directly leverages the user's role to provide a personalized onboarding experience.
  + **Role Impact:** customer, provider.

**Day 74-77: Performance Optimization**

* **Task Name:** Image Optimization & List Performance
  + **Description:** Improve the app's performance by optimizing image loading and list rendering.
  + **Implementation Details:** Use packages like cached\_network\_image to handle image caching. Implement lazy loading and pagination on long lists of services and providers.
  + **Synchronization Notes:** This is a general improvement that enhances the performance of all screens that display dynamic content, such as the home screen, service lists, and provider lists.
  + **Role Impact:** All roles.

**Week 12: Basic AI Features**

**Day 78-80: Simple Chatbot**

* **Task Name:** Rule-Based Chatbot
  + **Description:** Implement a simple chatbot to assist users with service discovery.
  + **Implementation Details:** Create a chat interface. The bot's logic is based on simple keyword matching. For example, if a user types "plumber," the bot suggests the "Plumbing" service category. This functionality is **only available to the customer role**.
  + **Synchronization Notes:** The chatbot's suggestions directly link to the service catalog from Week 3.
  + **Role Impact:** customer.
* **Task Name:** Service Recommendations
  + **Description:** Create a basic recommendation engine.
  + **Implementation Details:** Implement a simple algorithm on the backend that recommends services based on a provider's rating or popularity. The app will display these recommendations in a dedicated section on the home screen.
  + **Synchronization Notes:** This feature uses the ServiceProvider and Review models to provide initial recommendations to the customer role.
  + **Role Impact:** customer.

Here's the next phase of the SmartSheba implementation, focusing on safety, real-time tracking, and the critical **Admin Panel**.

**Phase 4: Safety & Admin Panel (Weeks 13-16)**

This phase adds a crucial layer of trust and security to the platform. It introduces emergency features for customers and builds the central admin dashboard for role management and platform oversight.

**Week 13: Emergency Features**

**Day 85-87: SOS System Foundation**

* **Task Name:** SOS Button & Alert System
  + **Description:** Implement a prominent SOS button for customers in case of an emergency.
  + **Implementation Details:** The SOS button is **only visible to the customer role**. When a user presses it, their location and identity are sent to the backend. A background service sends an alert via SMS (using a service like SSL Wireless) to pre-defined emergency contacts. The event is also logged in the database for admin review.
  + **Synchronization Notes:** This uses the UserEntity from Week 1 to identify the user and a backend API to trigger alerts, making it a powerful and secure feature.
  + **Role Impact:** customer, admin.

**Day 88-91: Safety Features**

* **Task Name:** Provider Verification Display
  + **Description:** Visually indicate that a provider's identity has been verified by an admin.
  + **Implementation Details:** When an admin approves a provider's application (Week 4), the backend sets a isVerified flag. A green badge is displayed on the provider's profile card and detail screen if this flag is true.
  + **Synchronization Notes:** This is a direct frontend representation of the backend data managed by the admin role.
  + **Role Impact:** customer, provider.
* **Task Name:** Incident Reporting
  + **Description:** Create a system for any user to report a safety concern.
  + **Implementation Details:** Implement a form where users can report issues. The report is submitted to the backend and sent to the admin dashboard for review.
  + **Synchronization Notes:** The report is tied to the UserEntity, ensuring the report is logged with the correct user ID.
  + **Role Impact:** All roles, with admin for review.

**Week 14: Real-Time Tracking**

**Day 92-94: Live Location Tracking**

* **Task Name:** Real-Time Location Sharing
  + **Description:** Allow a provider to share their live location with a customer during a service.
  + **Implementation Details:** The provider role can enable live location sharing from their dashboard for an active booking. The app sends location updates to the backend. The customer role can view this location on a map in their booking details.
  + **Synchronization Notes:** This feature directly links the provider and customer roles to a specific booking, ensuring a secure and contextual tracking experience.
  + **Role Impact:** customer, provider.

**Day 95-98: Advanced Tracking**

* **Task Name:** Geofencing
  + **Description:** Implement geofencing to trigger events based on a provider's location.
  + **Implementation Details:** Define a service area for the provider (from Week 8) and a geofence around the customer's location. When the provider enters or exits this geofence, a push notification is sent to the customer (e.g., "Your provider has arrived!").
  + **Synchronization Notes:** This combines location data, booking status, and the notification system to automate a key communication point.
  + **Role Impact:** customer, provider.

**Week 15: Payment Gateway Integration**

**Day 99-101: Mobile Banking Integration**

* **Task Name:** bKash/Nagad Integration
  + **Description:** Integrate with real mobile banking payment gateways.
  + **Implementation Details:** Use a local payment gateway's SDK (e.g., SSLCommerz, ShurjoPay) to handle bKash and Nagad payments. Implement the full transaction flow, including payment initiation and confirmation via a backend webhook. The payment screen is **only accessible to the customer role**.
  + **Synchronization Notes:** This is a crucial monetization layer that builds on the pricing system from Week 7.
  + **Role Impact:** customer.

**Day 102-105: Payment Processing & Refunds**

* **Task Name:** Refund System
  + **Description:** Create a system for customers to request refunds.
  + **Implementation Details:** A customer can submit a refund request for a completed service. This request is logged in the backend, but the final processing of the refund is an action that is **exclusive to the admin role**.
  + **Synchronization Notes:** This is a clear example of RBAC. The customer can initiate the request, but a platform admin must approve the final action.
  + **Role Impact:** customer, admin.

**Week 16: Admin Panel & Analytics**

**Day 106-108: Admin Panel Core**

* **Task Name:** Admin Authentication & Portal
  + **Description:** Develop the central platform management tool.
  + **Implementation Details:** Create a separate web or desktop application. All API endpoints for this portal are **protected by the admin or superAdmin roles**. Admins can view and manage all users.
  + **Synchronization Notes:** This is the culmination of the RBAC strategy. It is the gatekeeper for all platform-level changes.
  + **Role Impact:** admin, superAdmin.

**Day 109-112: Role & Verification Management**

* **Task Name:** Provider Approval Workflow
  + **Description:** Implement the process for admins to approve new provider applications.
  + **Implementation Details:** The Admin Panel will display a list of pending provider applications (from Week 4). An admin can review the submitted documents and **manually change the user's role from customer to provider**. This single action unlocks all provider-specific features for that user.
  + **Synchronization Notes:** This task completes the provider registration flow started in Week 4 and is a prime example of the power and security of our RBAC system.
  + **Role Impact:** admin, provider.

Here's the final phase of the SmartSheba implementation roadmap, completing the entire development cycle from a foundational platform to an AI-powered service.

**Phase 5: AI/ML Integration (Weeks 17-20)**

This phase introduces advanced features that leverage artificial intelligence to make the platform more intelligent and helpful for users.

**Week 17: Image Recognition**

**Day 113-115: Camera Integration & Image Upload**

* **Task Name:** Camera and Gallery Access
  + **Description:** Implement the functionality for users to upload images to the platform.
  + **Implementation Details:** Use the image\_picker package to allow users to take a photo or select one from their gallery. Implement a secure upload function that compresses images and uploads them to a cloud storage service. This feature is **only available to the customer role** within the booking or reporting flow.
  + **Synchronization Notes:** This uses the user's id from the UserEntity (Phase 1) to securely associate the uploaded image with the correct user.
  + **Role Impact:** customer.

**Day 116-119: AI Image Classification**

* **Task Name:** AI-Powered Service Suggestions
  + **Description:** Integrate a machine learning model to classify images and suggest relevant services.
  + **Implementation Details:** On the backend, run the uploaded image through a pre-trained **TensorFlow Lite model** to classify the problem (e.g., "plumbing issue," "electrical fault"). The app then receives the classification and suggests a relevant service category to the user. This functionality is **for the customer role** only.
  + **Synchronization Notes:** This feature directly uses the image upload functionality from the previous task. The service suggestions will link directly to the service catalog from Week 3.
  + **Role Impact:** customer.

**Week 18: Advanced AI Chatbot**

**Day 120-122: Natural Language Processing (NLP)**

* **Task Name:** Upgrade Chatbot Logic
  + **Description:** Enhance the chatbot from a rule-based system to one that understands user intent and context.
  + **Implementation Details:** Implement a backend service that uses a basic NLP model to process user messages. Instead of simple keyword matching, it should be able to identify user intent (e.g., "inquire about booking status") and extract entities (e.g., "plumber," "A/C repair"). This feature is **for the customer role**.
  + **Synchronization Notes:** This is a direct upgrade to the simple chatbot from Week 12. It uses the same user interface but a more sophisticated backend to provide a smarter user experience.
  + **Role Impact:** customer.

**Day 123-126: Contextual Chatbot Integration**

* **Task Name:** Multi-turn Conversations
  + **Description:** Create a chatbot that can maintain context across multiple messages.
  + **Implementation Details:** The chatbot should be able to follow a conversation flow and ask follow-up questions to clarify a user's request. This makes the chatbot a more effective tool for guiding customers to the right service.
  + **Synchronization Notes:** This builds on the NLP logic, allowing for a more human-like conversation that can lead directly to a booking or a service search.
  + **Role Impact:** customer.

**Week 19: Smart Recommendation System**

**Day 127-129: User Behavior Tracking**

* **Task Name:** Activity Logging
  + **Description:** Implement a system to log user activity for personalized recommendations.
  + **Implementation Details:** Create backend endpoints to track key user actions, such as service\_viewed, provider\_viewed, and booking\_completed. This data is associated with the user's ID and stored in an analytics database.
  + **Synchronization Notes:** This builds on the user management system from Phase 1. The data collected here is crucial for training the recommendation engine.
  + **Role Impact:** All roles.

**Day 130-133: Personalized Recommendations**

* **Task Name:** Recommendation Engine
  + **Description:** Develop an engine that provides personalized service and provider suggestions.
  + **Implementation Details:** Use the behavioral data from the previous task to create a basic recommendation algorithm. Recommendations can be based on a user's booking history, popular services in their area, or highly-rated providers. This feature is **for the customer role** only.
  + **Synchronization Notes:** This feature directly uses the data from the previous task and displays it within the CustomerDashboard from Week 2.
  + **Role Impact:** customer.

**Week 20: Final Polish & Production**

**Day 134-136: End-to-End Testing**

* **Task Name:** Comprehensive System Testing
  + **Description:** Conduct a final, comprehensive system test across all user roles.
  + **Implementation Details:** Run a full test suite to verify all user journeys, including login, booking, payment, and provider management. All **RBAC permissions** must be tested to ensure no unauthorized access.
  + **Synchronization Notes:** This is a crucial step that validates the integrity and security of the entire system built over the last 19 weeks.
  + **Role Impact:** All roles.

**Day 137-140: Production Launch Preparation**

* **Task Name:** App Store and Play Store Submission
  + **Description:** Prepare the app for final release.
  + **Implementation Details:** Configure production environment variables, API endpoints, and security keys. Prepare all necessary app store metadata, screenshots, and privacy policy for submission.
  + **Synchronization Notes:** This final task ensures the entire application is ready for deployment and public use.
  + **Role Impact:** All roles.