Integrations Sprint

Strategic Implementation Plan for Loma App

Current State Summary

- ✅ **Complete**: Onboarding flow (11 screens), main app navigation, UI/UX design
- ❌ **Missing**: All backend services, authentication, database, payments, AI integration
- ⚠️ **Critical Issues**: Plain text password storage, no data validation, client-side only

---

Recommended Implementation Order

PHASE 1: FOUNDATION - Supabase + Authentication (Priority: CRITICAL)

Timeline: Week 1-2 | Effort: 40-60 hours

Why This First:

1. **Security Critical**: Currently storing passwords in plain text - this is a major security vulnerability
2. **Legal Requirement**: Cannot accept payments without proper user accounts and data protection
3. **Foundation for Everything**: Auth is required before implementing payments, AI generation, or any server features
4. **Multi-device Support**: Users expect to access their data across devices

Implementation Steps:

1.1 Supabase Project Setup

- Create Supabase project and obtain credentials
- Design database schema:

```
- users (extends Supabase auth.users)
- user_profiles (onboarding data, preferences)
- recipes (AI-generated recipes)
- user_recipes (saved/favorited recipes)
- subscriptions (plan, tokens, status)
- progress_tracking (streaks, metrics)
```

- Configure Row Level Security (RLS) policies
- Set up Edge Functions for serverless backend logic

1.2 Authentication Service Implementation

- Install `@supabase/supabase-js`
- Create `/src/services/auth/supabase.ts` client configuration
- Create `/src/services/auth/authService.ts` with methods:
  - `signUp(email, password, userData)`
  - `signIn(email, password)`
  - `signOut()`
  - `resetPassword(email)`
  - `verifyEmail(token)`
- Replace UserContext auth logic with real Supabase Auth
- Implement session persistence and refresh

1.3 User Data Migration Strategy

- Modify UserContext to sync with Supabase
- Keep AsyncStorage as local cache (offline capability)
- Implement bi-directional sync strategy

- Handle data conflicts (last-write-wins or user prompt)
- Add migration flow for existing local users

1.4 Security Hardening

- Remove plain text password storage
- Implement JWT token handling
- Add request interceptors for auth headers
- Implement error handling for auth failures
- Add session timeout and refresh logic

Deliverables:

✅ Real user accounts with secure authentication

✅ Email verification system

✅ Password reset functionality

✅ User data synced to cloud database

✅ Multi-device support enabled

PHASE 2: PAYMENT INTEGRATION - Stripe (Priority: HIGH)

Timeline: Week 3 | Effort: 20-30 hours

Why After Phase 1:

1. **Requires Auth**: Cannot process payments without verified user accounts
2. **Compliance**: Need user database for subscription tracking and invoicing
3. **Token System**: Must track token balance server-side to prevent manipulation
4. **Legal Protection**: Proper auth provides audit trail for transactions

Implementation Steps:

2.1 Stripe Setup

- Create Stripe account (test + production modes)

- Install `@stripe/stripe-react-native`
- Create products and pricing in Stripe Dashboard:
  - Weekly: $3.99 (5 Munchies)
  - Monthly: $7.99 (20 Munchies)
  - Yearly: $48.99 (240 Munchies)
- Set up webhook endpoint (Supabase Edge Function)

2.2 Checkout Flow Implementation

- Update `PaymentScreen.tsx` to use real Stripe Checkout
- Implement payment sheet integration
- Handle payment success/failure
- Store subscription data in Supabase
- Generate receipt/invoice

2.3 Token/Credit System

- Add `tokens_balance` and `tokens_used` to user_profiles table
- Create Supabase Edge Function to validate token usage
- Implement token deduction on recipe generation (server-side)
- Add token purchase flow (top-up functionality)
- Display real-time token balance in UI

2.4 Subscription Management

- Implement webhook handlers:
  - `customer.subscription.created`
  - `customer.subscription.updated`
  - `customer.subscription.deleted`
  - `invoice.payment_succeeded`
  - `invoice.payment_failed`
- Update `SubscriptionScreen.tsx` with real data

- Implement plan changes and cancellations
- Add grace period handling for failed payments
- Implement billing history view

2.5 Platform-Specific Considerations

- iOS: Add in-app purchase support (Apple requirements)
- Android: Google Play billing integration
- Consider using RevenueCat for cross-platform IAP management

Deliverables:

✅ Real payment processing via Stripe

✅ Active subscription management

✅ Token system with server-side validation

✅ Billing history and invoices

✅ Webhook handling for subscription events

PHASE 3: AI RECIPE GENERATION (Priority: HIGH)

Timeline: Week 4-5 | Effort: 30-40 hours

Why After Phase 2:

1. **Most Expensive Feature**: AI API calls cost money - should only work for paid users
2. **Token Enforcement**: Requires payment system to track and limit usage
3. **Can Work Independently**: While payment system is being tested, can develop AI integration
4. **Revenue Protection**: Prevents abuse and ensures ROI on AI costs

Implementation Steps:

3.1 AI Provider Selection Choose based on requirements:

- **OpenAI GPT-4**: Best for recipe generation, widely used, good docs

- **Anthropic Claude**: Excellent at structured output, cost-effective
- **Custom Fine-tuned Model**: Best long-term cost but requires training

Recommendation: Start with OpenAI GPT-4 Turbo (cheaper, faster)

3.2 Recipe Generation Service

- Create `/src/services/recipes/aiService.ts`
- Implement server-side generation (Supabase Edge Function):

```
generateRecipe({
  mealType: 'breakfast' | 'lunch' | 'dinner' | 'snack',
  userPreferences: {
    dietaryRestrictions: [],
    allergens: [],
    cuisinePreferences: [],
    goals: [],
    equipment: []
  },
  customRequest?: string
})
```

- Design prompt engineering strategy:
  - System prompt with recipe format
  - User preferences injection
  - Structured JSON output
  - Nutrition calculation logic
- Implement response parsing and validation
- Handle API errors and retries

3.3 Integration with Existing Flow

- Update `HomeScreen.tsx`:
  - Replace mock generation with API call
  - Add loading state with progress indicator

    - Handle generation errors gracefully
- Update `RecipeGeneratedScreen.tsx`:
    - Show AI-generated options (generate 4 variants)
    - Allow regeneration (costs tokens)
- Update `RecipeReviewScreen.tsx`:
    - Save to Supabase `recipes` table
    - Link to user in `user_recipes` table
    - Update token balance

3.4 Recipe Database

- Implement recipe CRUD operations in Supabase
- Add recipe versioning (track edits)
- Implement recipe search/filtering
- Add recipe sharing capability (optional MVP+)
- Sync RecipeContext with Supabase

3.5 Personalization Engine

- Use user's onboarding data for personalization
- Track recipe history for preference learning
- Implement recipe recommendation algorithm
- A/B test different prompt strategies

Deliverables:

✅ Real AI-powered recipe generation

✅ Personalized recipes based on user preferences

✅ Token deduction per generation

✅ Recipe storage in cloud database

✅ Recipe library synced across devices

PHASE 4: DATA PERSISTENCE & PROGRESS TRACKING (Priority: MEDIUM)

Timeline: Week 6 | Effort: 20-30 hours

Why Last:

1. **Non-Blocking**: App functions without real progress tracking
2. **Requires Usage Data**: Need real recipe generations and cooking sessions first
3. **Enhancement Feature**: Improves UX but not critical for launch
4. **Time to Collect Data**: Need users actively using the app to calculate meaningful metrics

Implementation Steps:

4.1 Real Progress Tracking

- Create `progress_tracking` table in Supabase
- Implement streak calculation algorithm:
  - Track daily recipe generations
  - Calculate current and longest streak
  - Handle timezone considerations
- Real-time streak updates on recipe generation
- Push notifications for streak maintenance

4.2 Cooking Session Tracking

- Add "Mark as Cooked" functionality to RecipeDetailScreen
- Track cooking sessions with timestamps
- Calculate real hours saved (based on recipe time)
- Calculate money saved (based on estimated cost vs eating out)
- Update ProgressScreen with real data

4.3 Achievement System

- Move achievement calculations server-side

- Implement achievement unlock logic
- Add push notifications for achievements
- Store achievement progress in database
- Real-time achievement sync

4.4 Analytics & Insights

- Track user engagement metrics
- Generate weekly summaries
- Implement Weekly Check-in with real data
- Add nutrition tracking over time
- Personal records and milestones

Deliverables:

✅ Real streak tracking with calculations

✅ Cooking session tracking

✅ Accurate hours/money saved metrics

✅ Achievement system with real progress

✅ Weekly insights and summaries

Pre-Implementation Checklist

Before Starting Phase 1:

☐ Create Supabase account and project

☐ Set up development environment (.env files)

☐ Install required dependencies:

```
npm install @supabase/supabase-js
npm install @stripe/stripe-react-native
npm install react-native-dotenv
```

- [ ] Create `/src/services` directory structure
- [ ] Set up error tracking (Sentry or similar)
- [ ] Plan data migration strategy for existing users

Technical Debt to Address:

1. **Create Service Layer Architecture**:

```
/src/services
  /auth        (authentication)
  /recipes     (AI + CRUD)
  /payments    (Stripe)
  /storage     (AsyncStorage + Supabase sync)
  /analytics   (tracking)
```

2. **Environment Configuration**:
   - Create `.env.development` and `.env.production`
   - Never commit secrets to git
   - Use Expo SecureStore for sensitive data
3. **Error Handling**:
   - Implement global error boundary
   - Add network error handling
   - User-friendly error messages
   - Offline mode support
4. **Data Validation**:
   - Add schema validation (Zod or Yup)
   - Validate API responses
   - Handle version mismatches

Risk Mitigation Strategies

High-Risk Areas:

1. **Payment Integration**: Test thoroughly in Stripe test mode, handle all edge cases
2. **Data Migration**: Provide rollback strategy if sync fails
3. **AI Costs**: Implement rate limiting and cost monitoring
4. **User Acquisition**: Have working product before marketing spend

Contingency Plans:

- **AI Costs Too High**: Implement caching, reduce generations, or increase prices
- **Supabase Limits**: Plan for scaling (upgrade tier or self-host)
- **Payment Failures**: Grace period + retry logic + support system
- **User Data Loss**: Regular backups + export functionality

Success Metrics

Phase 1 Success Criteria:

- [ ] 100% of users can sign up and log in
- [ ] Zero plain text passwords in storage
- [ ] User data syncs within 2 seconds
- [ ] Session persists across app restarts

Phase 2 Success Criteria:

- [ ] Payment success rate > 95%
- [ ] Token balance updates in real-time
- [ ] Webhook delivery rate > 99%
- [ ] Subscription management functional

Phase 3 Success Criteria:

- [ ] Recipe generation completes in < 10 seconds

- [ ] Generated recipes match user preferences
- [ ] Recipe quality rating > 4/5 stars
- [ ] API error rate < 1%

Phase 4 Success Criteria:

- [ ] Streak accuracy: 100%
- [ ] Progress metrics update within 5 seconds
- [ ] Achievement unlock rate > 50% of users
- [ ] User engagement increases 20%+

Cost Estimates (Monthly, at scale)

Infrastructure:

- **Supabase**: $0 (free tier) → $25/mo (pro) → $599/mo (team)
- **Stripe**: 2.9% + $0.30 per transaction
- **OpenAI API**: ~$0.02 per recipe generation (GPT-4 Turbo)
  - 1,000 users × 10 recipes/mo = $200/mo
  - Can optimize with caching and prompt engineering
- **Hosting**: Expo (free) or custom ($50-200/mo)
- **Error Tracking**: Sentry (free → $26/mo)

Break-Even Analysis:

- At $7.99/mo subscription
- Need ~250 paying users to break even with infrastructure
- Additional margin for development costs and growth

Final Recommendation

Immediate Next Steps (Do This First):

1. **Set Up Development Environment** (1-2 hours):
   - Create Supabase project
   - Install dependencies
   - Configure environment variables
   - Test Supabase connection
2. **Design Database Schema** (2-4 hours):
   - Map all UserContext fields to tables
   - Design relationships
   - Plan RLS policies
   - Document schema
3. **Create Service Layer Structure** (1-2 hours):
   - Set up `/src/services` directories
   - Create placeholder files
   - Define interfaces/types
4. **Begin Phase 1** (Authentication):
   - Start with Supabase Auth integration
   - Replace mock auth incrementally
   - Test thoroughly before moving on

Why This Approach Works:

✅ **Security First**: Fixes critical vulnerabilities immediately

✅ **Incremental**: Each phase builds on the previous

✅ **Revenue-Focused**: Payments come early, enabling monetization

✅ **Cost-Conscious**: Most expensive feature (AI) comes after revenue

✅ **User-Centric**: Core value (recipe generation) prioritized

✅ **Testable**: Each phase has clear success criteria

Timeline Summary:

- **Week 1-2**: Authentication + Database (FOUNDATION)
- **Week 3**: Payments + Subscriptions (MONETIZATION)
- **Week 4-5**: AI Recipe Generation (CORE VALUE)
- **Week 6**: Progress Tracking (ENGAGEMENT)

Total: 6 weeks to fully functional MVP with all core features

Would you like me to proceed with implementing any specific phase, or do you need more details on any aspect of this plan?