SCLA (Skiin Connected Life App) - Complete Reconstruction Documentation

Executive Summary

This document provides a comprehensive analysis and reconstruction guide for the SCLA (Skiin Connected Life App) mobile application based on detailed screen analysis. The app is a health monitoring platform that integrates with SKIIN wearable devices to collect ECG data, track symptoms, and provide clinical insights for Holter studies.

Key Features

- **Real-time ECG monitoring** with dual-channel display
- **Symptom logging** with intensity tracking and trigger identification
- **Blood pressure recording** with dual measurement capability
- **Diary functionality** for historical data review
- **Device management** for SKIIN pod pairing and monitoring
- **Clinical integration** for Holter study progress tracking

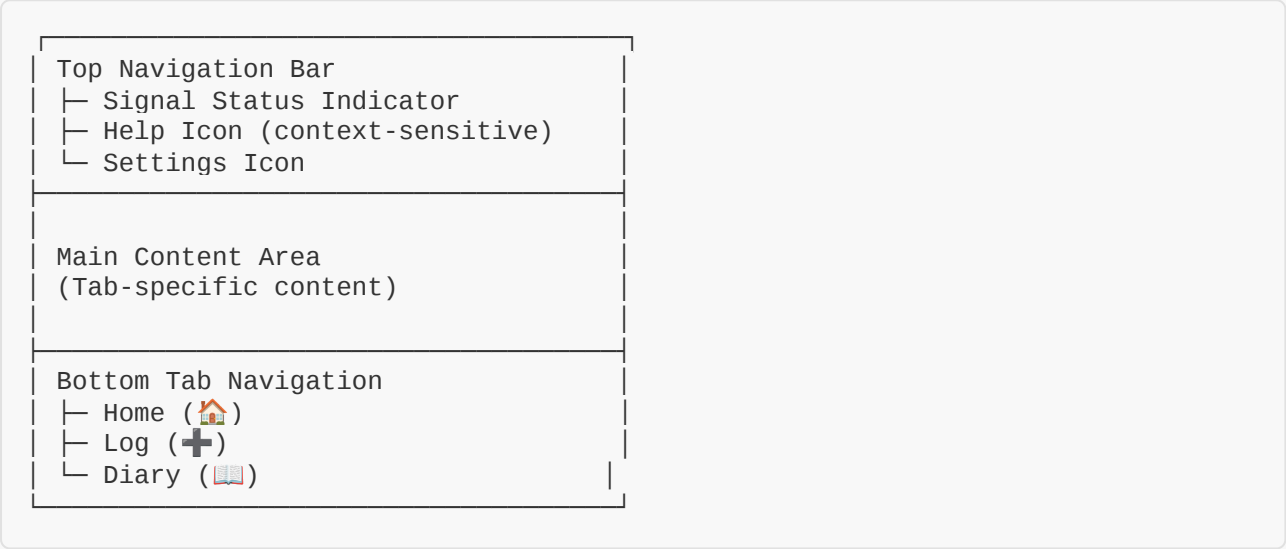
Table of Contents

7. Technical Implementation Guide
8. User Experience Guidelines

App Architecture Overview

The SCLA app follows a tab-based navigation pattern with three main sections:

Primary Navigation Structure



Core Modules

1. **Authentication Module**: User sign-in with multiple methods
2. **Device Management Module**: SKIIN pod pairing and monitoring
3. **Health Data Module**: ECG, symptoms, and blood pressure tracking
4. **Analytics Module**: Data analysis and clinical insights
5. **Settings Module**: User preferences and device configuration

Screen-by-Screen Analysis

Authentication Screens

Welcome Screen

- **Purpose**: App introduction and entry point
- **Key Elements**:
- SKIIN branding and feature overview
- "Get Started" button for new users
- "Sign In" link for existing users
- **Navigation**: Entry point to registration or sign-in flows

Sign In Screen

- **Purpose**: User authentication with multiple options
- **Key Elements**:
- Email/password fields with show/hide toggle
- "Forgot Password" link
- Alternative authentication methods (6-digit code, QR code)
- "Sign Up" link for new users
- **Validation**: Email format validation, password requirements
- **Security**: Secure credential handling, session management

Device Pairing Screens

Device Pairing Introduction

- **Purpose**: Guide users through pod pairing process
- **Key Elements**:
- Visual instructions with device imagery
- Step-by-step pairing guidance
- "Pair Pod" and "Cancel" actions

- **Hardware Integration**: Bluetooth Low Energy (BLE) connectivity

Pod Selection Screen

- **Purpose**: Select specific device from available options
- **Key Elements**:
- List of discoverable devices with serial numbers
- Radio button selection interface
- Connection strength indicators
- **Technical Requirements**: BLE device discovery and filtering

Connection Status Screen

- **Purpose**: Provide feedback during pairing process
- **Key Elements**:
- Progress indication
- Device information display
- Connection status updates
- **User Experience**: Clear feedback and error handling

Main Application Screens

Home Dashboard

- **Purpose**: Central hub for status overview and quick access
- **Key Components**:
- Greeting message with time-based personalization
- Garment Signal Status card with real-time updates
- Holter Study Progress tracking
- 14-Day Progress visualization
- Battery optimization prompts
- **Real-time Features**: Live signal status, sync indicators
- **Responsive Design**: Adaptive layout for different screen sizes

ECG Viewer (Signal Status Screen)

- **Purpose**: Real-time ECG monitoring and signal quality assessment
- **Technical Specifications**:
- Dual-channel ECG display (Channel 1 and Channel 3)
- Configurable scale settings (mm/mV)
- Real-time waveform rendering at 250Hz+ sample rate
- Signal quality indicators with color coding
- Heart rate calculation and display
- **User Interactions**:
- Scale adjustment dropdown
- Signal troubleshooting guidance
- Navigation back to dashboard
- **Performance Requirements**: Low-latency data streaming, smooth animations

Log Tab Interface

- **Purpose**: Data entry hub for health metrics
- **Modal Design**: "What would you like to add?" selection
- **Entry Types**:
- Symptom logging with detailed attributes
- Blood pressure recording with dual measurements
- **User Experience**: Quick access, minimal friction data entry

Symptom Logging Flow

Basic Symptom Entry

- **Date/Time Selection**: Calendar and time picker components
- **Symptom Selection**: Grid layout with predefined options
- **Search Functionality**: Real-time filtering of symptom list
- **Custom Symptoms**: Ability to add user-defined symptoms

Detailed Symptom Entry

- **Intensity Rating**: 0-10 slider with visual feedback
- **Trigger Selection**: Multi-select with predefined and custom options
- **Duration Tracking**: Ongoing vs. intermittent with time inputs
- **Notes Field**: Free-text additional details
- **Validation**: Required field checking, data format validation

Symptom Confirmation

- **Success Feedback**: Visual confirmation with illustration
- **ECG Analysis Integration**: Automatic correlation with ECG data
- **Analysis Timeline**: Expected completion time communication
- **Return Navigation**: Clear path back to main interface

Blood Pressure Logging Flow

Measurement Entry

- **Dual Readings**: First and second measurement capability
- **Numeric Input**: Custom keypad for precise entry
- **Measurement Guidelines**: Visual instructions for proper technique
- **Data Validation**: Range checking, format validation

Confirmation and Storage

- **Success Confirmation**: Visual feedback for completed entry
- **Data Integration**: Automatic sync with health record
- **Historical Tracking**: Integration with diary timeline

Diary Interface

Calendar View

- **Monthly Navigation**: Month/year selection dropdown

- **Date Selection**: Interactive calendar with entry indicators
- **Entry Timeline**: Chronological list of daily entries
- **Refresh Capability**: Pull-to-refresh for data updates

Entry Display

- **Card-Based Layout**: Distinct cards for different data types
- **Timestamp Precision**: Exact time display for each entry
- **Status Indicators**: Analysis and review status badges
- **Quick Actions**: Edit and delete functionality

Entry Detail View

- **Comprehensive Display**: All entry attributes visible
- **Edit Functionality**: In-place editing capability
- **Remove Options**: Confirmation-based deletion
- **Analysis Status**: Pending/completed analysis indicators

Settings and Management

Device Management

- **Connection Status**: Real-time device connectivity
- **Sync Information**: Last sync time and current status
- **Battery Monitoring**: Device charge level display
- **Device Details**: Firmware, hardware version information
- **Unpair Functionality**: Device removal capability

Advanced Settings

- **User Profile**: Personal information management
- **Clinical Program**: Study-specific configurations
- **Data Upload Preferences**: WiFi vs. cellular options
- **Custom Content Management**: Symptoms and triggers

Manage Symptoms & Triggers

- **Custom Symptoms**: User-created symptom list
- **Custom Triggers**: User-defined trigger management
- **Delete Functionality**: Remove custom entries
- **Sync Integration**: Cloud synchronization of custom data

Navigation Flows

Primary User Journeys

First-Time User Flow

```
Welcome Screen → Sign Up → Device Pairing → Tutorial → Home Dashboard
```

Returning User Flow

```
Sign In → Home Dashboard → [Daily Usage Patterns]
```

Symptom Logging Journey

```
Home/Log Tab → Add Data Modal → Symptom Entry → Details Form → Confirmation →
Diary Update
```

ECG Monitoring Journey

```
Home Dashboard → Signal Status → ECG Viewer → Troubleshooting (if needed)
```

Data Review Journey

```
Diary Tab → Calendar Navigation → Entry Selection → Detail View → Edit/Remove
```

Modal and Popup Interactions

Battery Optimization Popup

- **Trigger**: System detection of suboptimal settings
- **Options**: Accept optimization or dismiss
- **Integration**: Direct link to device settings

Signal Quality Alerts

- **Trigger**: Poor ECG signal detection
- **Actions**: Troubleshooting guidance or dismissal
- **Context**: Real-time signal monitoring

Add Data Modal

- **Trigger**: Log tab access or plus button
- **Options**: Symptom or blood pressure entry
- **Design**: Bottom sheet modal with clear options

Component Hierarchy

UI Component Structure

Navigation Components

- **BottomTabNavigator**: Main app navigation
- **TopNavigationBar**: Context-sensitive header
- **SideNavigationMenu**: Settings and profile access
- **BackButton**: Consistent navigation pattern

Data Display Components

- **ECGViewer**: Real-time waveform display
- **SignalStatusIndicator**: Connection and quality status

- **EntryCard**: Diary entry display component
- **ProgressIndicator**: Study and sync progress
- **CalendarComponent**: Date selection and navigation

Input Components

- **SymptomSelector**: Grid-based symptom selection
- **IntensitySlider**: 0-10 rating scale
- **TriggerSelector**: Multi-select trigger interface
- **DateTimePicker**: Date and time selection
- **NumericKeypad**: Custom numeric input
- **SearchInput**: Real-time filtering capability

Modal Components

- **AddDataModal**: Entry type selection
- **ConfirmationModal**: Success feedback
- **AlertModal**: Error and warning messages
- **SettingsModal**: Configuration interfaces

State Management Architecture

Global State

- **User Authentication**: Login status, user profile
- **Device Connection**: Pairing status, sync state
- **Real-time Data**: ECG stream, signal quality
- **Application Settings**: Preferences, configurations

Local State

- **Form Data**: Entry forms, validation states
- **UI State**: Modal visibility, navigation state
- **Cache Data**: Offline storage, sync queue

- **Temporary Data**: Search results, filtered lists

Data Models

Core Entity Relationships

```
 User (1) ⟷ (M) UserDevices ⟷ (M) Devices
User (1) ⟷ (M) Symptoms
User (1) ⟷ (M) BloodPressureReadings
User (1) ⟷ (M) ECGData
Symptoms (1) ⟷ (M) SymptomTriggers
User (1) ⟷ (M) CustomSymptoms
User (1) ⟷ (M) CustomTriggers
User (1) ⟷ (1) UserPreferences
User (1) ⟷ (M) HolterStudies
ECGData (1) ⟷ (M) ECGAnalyses
```

Key Data Structures

User Profile

```typescript
interface User {
  id: string;
  email: string;
  firstName: string;
  lastName: string;
  dateOfBirth: Date;
  createdAt: Date;
  lastLogin: Date;
  isActive: boolean;
}
```

Device Information

```typescript
interface Device {
  id: string;
  serialNumber: string;
  deviceType: 'skiin_pod' | 'chestband';
  firmwareVersion: string;
  hardwareVersion: string;
  batteryLevel: number; // 0-100
  connectionStatus: 'connected' | 'disconnected' | 'syncing';
  lastSyncAt: Date;
}
```

Symptom Entry

```typescript
interface Symptom {
  id: string;
  userId: string;
  symptomName: string;
  intensity: number; // 0-10
  experiencedAt: Date;
  durationType: 'ongoing' | 'intermittent';
  durationHours?: number;
  durationMinutes?: number;
  triggers: string[];
  notes?: string;
  analysisStatus: 'pending' | 'analyzing' | 'completed';
  reviewStatus: 'pending' | 'reviewed' | 'approved';
}
```

ECG Data

```typescript
interface ECGData {
  id: string;
  userId: string;
  deviceId: string;
  recordedAt: Date;
  durationSeconds: number;
  sampleRate: number; // Hz
  channel1Data: number[]; // Voltage values
  channel3Data: number[];
  signalQuality: {
    channel1: 'good' | 'poor' | 'no_signal';
    channel3: 'good' | 'poor' | 'no_signal';
  };
  heartRate?: number;
  analysisStatus: 'pending' | 'analyzing' | 'completed' | 'failed';
}
```

Blood Pressure Reading

```typescript
interface BloodPressureReading {
  id: string;
  userId: string;
  measuredAt: Date;
  systolic1: number;
  diastolic1: number;
  systolic2?: number;
  diastolic2?: number;
  notes?: string;
}
```

API Specifications

Authentication Endpoints

POST /api/auth/login

Request:

```
{
  "email": "user@example.com",
  "password": "securePassword123"
}
```

Response:

```
{
  "accessToken": "eyJhbGciOiJIUzI1NiIs...",
  "refreshToken": "eyJhbGciOiJIUzI1NiIs...",
  "user": {
    "id": "550e8400-e29b-41d4-a716-446655440000",
    "email": "user@example.com",
    "firstName": "John",
    "lastName": "Doe"
  },
  "expiresIn": 900
}
```

POST /api/auth/login/qr

Request:

```
{
  "qrCode": "..."
}
```

POST /api/auth/login/six-digit

Request:

```
{
  "code": "123456",
  "deviceId": "device_identifier"
}
```

Device Management Endpoints

GET /api/devices/discover

Response:

```
{
  "devices": [
    {
      "serialNumber": "31067601890",
      "deviceType": "skiin_pod",
      "signalStrength": -45,
      "isAvailable": true
    },
    {
      "serialNumber": "31046900579",
      "deviceType": "skiin_pod",
      "signalStrength": -62,
      "isAvailable": true
    }
  ]
}
```

POST /api/devices/pair

Request:

```
{
  "serialNumber": "31067601890",
  "deviceName": "My Skiin Pod"
}
```

Response:

```
{
  "device": {
    "id": "550e8400-e29b-41d4-a716-446655440000",
    "serialNumber": "31067601890",
    "deviceName": "My Skiin Pod",
    "pairingStatus": "success",
    "firmwareVersion": "18.0.9.1",
    "hardwareVersion": "20"
  }
}
```

GET /api/devices/status

Response:

```
{
  "devices": [
    {
      "id": "550e8400-e29b-41d4-a716-446655440000",
      "serialNumber": "31067601890",
      "connectionStatus": "connected",
      "batteryLevel": 75,
      "lastSyncAt": "2025-06-24T11:12:00Z",
      "syncStatus": "completed",
      "signalQuality": {
        "channel1": "good",
        "channel3": "poor"
      }
    }
  ]
}
```

Health Data Endpoints

GET /api/ecg/realtime

WebSocket Connection:

```
wss://api.scla.com/ws/ecg/{userId}
Authorization: Bearer {accessToken}
```

Data Stream Format:

```
{
  "timestamp": "2025-06-24T11:14:30.123Z",
  "deviceId": "550e8400-e29b-41d4-a716-446655440000",
  "sampleRate": 250,
  "channel1": [0.1, 0.15, 0.12, 0.08, ...],
  "channel3": [0.05, 0.08, 0.06, 0.04, ...],
  "heartRate": 72,
  "signalQuality": {
    "channel1": "good",
    "channel3": "poor"
  }
}
```

POST /api/symptoms

Request:

```
{
  "symptomName": "Brain Fog",
  "intensity": 5,
  "experiencedAt": "2025-06-24T11:13:00Z",
  "durationType": "ongoing",
  "durationHours": 2,
  "durationMinutes": 30,
  "triggers": ["Caffeine", "Not Sure", "hot"],
  "notes": "Felt foggy after morning coffee"
}
```

Response:

```
{
  "symptom": {
    "id": "550e8400-e29b-41d4-a716-446655440000",
    "symptomName": "Brain Fog",
    "intensity": 5,
    "experiencedAt": "2025-06-24T11:13:00Z",
    "analysisStatus": "pending",
    "ecgCorrelationId": "550e8400-e29b-41d4-a716-446655440001"
  }
}
```

GET /api/symptoms

Query Parameters: - `startDate` : ISO 8601 date string - `endDate` : ISO 8601 date string - `page` : Page number (default: 1) - `limit` : Items per page (default: 20)

Response:

```
{
  "symptoms": [
    {
      "id": "550e8400-e29b-41d4-a716-446655440000",
      "symptomName": "Brain Fog",
      "intensity": 5,
      "experiencedAt": "2025-06-24T11:13:00Z",
      "triggers": ["Caffeine", "Not Sure", "hot"],
      "analysisStatus": "completed",
      "reviewStatus": "pending"
    }
  ],
  "pagination": {
    "page": 1,
    "limit": 20,
    "total": 45,
    "totalPages": 3
  }
}
```

POST /api/blood-pressure

Request:

```
{
  "measuredAt": "2025-06-24T11:12:00Z",
  "systolic1": 128,
  "diastolic1": 84,
  "systolic2": 129,
  "diastolic2": 85,
  "notes": "Morning reading after coffee"
}
```

Diary and Analytics Endpoints

GET /api/diary

Query Parameters: - `date`: YYYY-MM-DD format - `month`: YYYY-MM format - `year`: YYYY format

Response:

```
{
  "date": "2025-06-24",
  "entries": [
    {
      "id": "550e8400-e29b-41d4-a716-446655440000",
      "type": "symptom",
      "timestamp": "2025-06-24T10:38:00Z",
      "data": {
        "symptomName": "Sweating",
        "intensity": 5,
        "triggers": ["Caffeine"],
        "analysisStatus": "pending",
        "reviewStatus": "pending"
      }
    },
    {
      "id": "550e8400-e29b-41d4-a716-446655440001",
      "type": "blood_pressure",
      "timestamp": "2025-06-24T11:12:00Z",
      "data": {
        "systolic": 128.5,
        "diastolic": 84,
        "reading": "128.5/84 mmHg"
      }
    }
  ]
}
```

GET /api/holter-study/progress

Response:

```
{
  "study": {
    "id": "550e8400-e29b-41d4-a716-446655440000",
    "name": "14-Day Holter Study",
    "startDate": "2025-06-10",
    "endDate": "2025-06-24",
    "progressPercentage": 65,
    "daysCompleted": 9,
    "totalDays": 14,
    "status": "active",
    "lastUpdated": "2025-06-24T10:21:00Z"
  }
}
```

Settings and Preferences Endpoints

GET /api/user/preferences

Response:

```
{
  "preferences": {
    "dataUploadPreference": "wifi_cellular",
    "notificationEnabled": true,
    "ecgScaleSetting": 10.0,
    "timezone": "America/New_York",
    "language": "en"
  }
}
```

PUT /api/user/preferences

Request:

```
{
  "dataUploadPreference": "wifi_only",
  "notificationEnabled": false,
  "ecgScaleSetting": 5.0
}
```

GET /api/custom-symptoms

Response:

```
{
  "customSymptoms": [
    {
      "id": "550e8400-e29b-41d4-a716-446655440000",
      "name": "Custom Headache Type",
      "createdAt": "2025-06-20T14:30:00Z"
    }
  ]
}
```

POST /api/custom-triggers

Request:

```
{
  "triggerName": "hot weather"
}
```

DELETE /api/custom-triggers/{triggerId}

Technical Implementation Guide

Mobile App Architecture

Technology Stack Recommendations

- **Framework**: React Native or Flutter for cross-platform development
- **State Management**: Redux Toolkit or MobX for complex state
- **Navigation**: React Navigation or Flutter Navigator
- **Real-time Communication**: WebSocket with Socket.io
- **Local Storage**: SQLite with encryption
- **Charts/Graphs**: Victory Native or FL Chart
- **Bluetooth**: React Native BLE Manager or Flutter Blue

Project Structure

```
src/
├── components/           # Reusable UI components
│   ├── common/          # Generic components
│   ├── ecg/             # ECG-specific components
│   ├── forms/           # Form components
│   └── navigation/      # Navigation components
├── screens/             # Screen components
│   ├── auth/            # Authentication screens
│   ├── home/            # Home dashboard
│   ├── log/             # Data entry screens
│   ├── diary/           # Diary and history
│   └── settings/        # Settings and preferences
├── services/            # API and business logic
│   ├── api/             # API client and endpoints
│   ├── bluetooth/       # Device communication
│   ├── storage/         # Local data management
│   └── analytics/       # Data analysis services
├── store/               # State management
│   ├── slices/          # Redux slices or MobX stores
│   └── middleware/      # Custom middleware
├── utils/               # Utility functions
│   ├── validation/      # Form validation
│   ├── formatting/      # Data formatting
│   └── constants/       # App constants
└── assets/              # Images, fonts, etc.
```

Key Implementation Considerations

Real-time ECG Display

```
// ECG Viewer Component Implementation
interface ECGViewerProps {
  deviceId: string;
  scaleSetting: number;
  onSignalQualityChange: (quality: SignalQuality) => void;
}

const ECGViewer: React.FC<ECGViewerProps> = ({
  deviceId,
  scaleSetting,
  onSignalQualityChange
}) => {
  const [ecgData, setEcgData] = useState<ECGDataPoint[]>([]);
  const [signalQuality, setSignalQuality] = useState<SignalQuality>();

  useEffect(() => {
    const websocket = new WebSocket(`wss://api.scla.com/ws/ecg/${deviceId}`);

    websocket.onmessage = (event) => {
      const data = JSON.parse(event.data);
      setEcgData(prev => [...prev.slice(-1000), ...data.samples]);
      setSignalQuality(data.signalQuality);
      onSignalQualityChange(data.signalQuality);
    };

    return () => websocket.close();
  }, [deviceId]);

  return (
    <View style={styles.container}>
      <ECGChart
        data={ecgData}
        scale={scaleSetting}
        signalQuality={signalQuality}
      />
      <SignalQualityIndicator quality={signalQuality} />
    </View>
  );
};
```

Bluetooth Device Management

```
// Device Service Implementation
class DeviceService {
  private bleManager: BleManager;

  async discoverDevices(): Promise<Device[]> {
    const devices = await this.bleManager.startDeviceScan(
      ['SKIIN_SERVICE_UUID'],
      { allowDuplicates: false }
    );

    return devices.filter(device =>
      device.name?.includes('SKIIN') &&
      device.isConnectable
    );
  }

  async pairDevice(serialNumber: string): Promise<PairingResult> {
    try {
      const device = await this.bleManager.connectToDevice(serialNumber);
      await device.discoverAllServicesAndCharacteristics();

      // Setup ECG data streaming
      await this.setupECGStreaming(device);

      return { success: true, device };
    } catch (error) {
      return { success: false, error: error.message };
    }
  }

  private async setupECGStreaming(device: Device): Promise<void> {
    await device.monitorCharacteristicForService(
      'ECG_SERVICE_UUID',
      'ECG_DATA_CHARACTERISTIC_UUID',
      (error, characteristic) => {
        if (characteristic?.value) {
          const ecgData = this.parseECGData(characteristic.value);
          this.onECGDataReceived(ecgData);
        }
      }
    );
  }
}
```

Form Validation and State Management

```
// Symptom Form Implementation
interface SymptomFormState {
  symptomName: string;
  intensity: number;
  experiencedAt: Date;
  triggers: string[];
  durationType: 'ongoing' | 'intermittent';
  durationHours?: number;
  durationMinutes?: number;
  notes: string;
}

const useSymptomForm = () => {
  const [formState, setFormState] = useState<SymptomFormState>({
    symptomName: '',
    intensity: 0,
    experiencedAt: new Date(),
    triggers: [],
    durationType: 'ongoing',
    notes: ''
  });

  const [errors, setErrors] = useState<Record<string, string>>({});

  const validateForm = (): boolean => {
    const newErrors: Record<string, string> = {};

    if (!formState.symptomName.trim()) {
      newErrors.symptomName = 'Symptom name is required';
    }

    if (formState.intensity < 0 || formState.intensity > 10) {
      newErrors.intensity = 'Intensity must be between 0 and 10';
    }

    if (formState.durationType === 'intermittent') {
      if (!formState.durationHours && !formState.durationMinutes) {
        newErrors.duration = 'Duration is required for intermittent symptoms';
      }
    }

    setErrors(newErrors);
    return Object.keys(newErrors).length === 0;
  };

  const submitForm = async (): Promise<boolean> => {
    if (!validateForm()) return false;

    try {
      await apiClient.post('/api/symptoms', formState);
      return true;
    } catch (error) {
      setErrors({ submit: 'Failed to save symptom' });
      return false;
    }
  };

  return {
    formState,
```

```
    setFormState,
    errors,
    validateForm,
    submitForm
  };
};
```

Backend Implementation

Technology Stack Recommendations

- **Runtime**: Node.js with TypeScript or Python with FastAPI
- **Database**: PostgreSQL with TimescaleDB for time-series data
- **Real-time**: Socket.io or WebSocket with Redis for scaling
- **Authentication**: JWT with refresh tokens
- **File Storage**: AWS S3 or Google Cloud Storage for ECG data
- **Message Queue**: Redis or RabbitMQ for analysis jobs
- **Monitoring**: Prometheus with Grafana
- **Deployment**: Docker with Kubernetes

Database Optimization

Time-Series Data Handling

```sql
 -- TimescaleDB hypertable for ECG data
CREATE TABLE ecg_data (
    id UUID DEFAULT gen_random_uuid(),
    user_id UUID NOT NULL,
    device_id UUID NOT NULL,
    recorded_at TIMESTAMPTZ NOT NULL,
    channel_1_data BYTEA,
    channel_3_data BYTEA,
    heart_rate INTEGER,
    signal_quality JSONB,
    PRIMARY KEY (id, recorded_at)
);

-- Convert to hypertable for time-series optimization
SELECT create_hypertable('ecg_data', 'recorded_at');

-- Create indexes for common queries
CREATE INDEX idx_ecg_user_time ON ecg_data (user_id, recorded_at DESC);
CREATE INDEX idx_ecg_device_time ON ecg_data (device_id, recorded_at DESC);
```

Data Retention Policies

```
-- Automatic data retention for ECG data
SELECT add_retention_policy('ecg_data', INTERVAL '2 years');

-- Compression for older data
SELECT add_compression_policy('ecg_data', INTERVAL '30 days');
```

API Implementation Examples

Real-time ECG Streaming

```
// WebSocket handler for ECG streaming
class ECGStreamHandler {
  private connectedClients = new Map<string, WebSocket>();

  handleConnection(ws: WebSocket, userId: string) {
    this.connectedClients.set(userId, ws);

    ws.on('close', () => {
      this.connectedClients.delete(userId);
    });

    // Send initial signal status
    this.sendSignalStatus(userId);
  }

  async broadcastECGData(userId: string, ecgData: ECGDataPoint) {
    const client = this.connectedClients.get(userId);
    if (client && client.readyState === WebSocket.OPEN) {
      client.send(JSON.stringify({
        type: 'ecg_data',
        data: ecgData,
        timestamp: new Date().toISOString()
      }));
    }
  }

  async sendSignalStatus(userId: string) {
    const status = await this.getLatestSignalStatus(userId);
    const client = this.connectedClients.get(userId);

    if (client && client.readyState === WebSocket.OPEN) {
      client.send(JSON.stringify({
        type: 'signal_status',
        data: status
      }));
    }
  }
}
```

Symptom Analysis Service

```
// Symptom analysis and ECG correlation
class SymptomAnalysisService {
  async analyzeSymptom(symptomId: string): Promise<AnalysisResult> {
    const symptom = await this.getSymptom(symptomId);
    const ecgData = await this.getECGDataAroundTime(
      symptom.userId,
      symptom.experiencedAt,
      { beforeMinutes: 30, afterMinutes: 30 }
    );

    if (!ecgData.length) {
      return { status: 'no_ecg_data', confidence: 0 };
    }

    // Perform analysis
    const analysis = await this.performECGAnalysis(ecgData, symptom);

    // Store results
    await this.storeAnalysisResults(symptomId, analysis);

    // Notify user if significant findings
    if (analysis.confidence > 0.8) {
      await this.notifyUser(symptom.userId, analysis);
    }

    return analysis;
  }

  private async performECGAnalysis(
    ecgData: ECGDataPoint[],
    symptom: Symptom
  ): Promise<AnalysisResult> {
    // Implement ECG analysis algorithms
    // This would typically involve:
    // 1. Heart rate variability analysis
    // 2. Arrhythmia detection
    // 3. QT interval measurement
    // 4. Correlation with symptom timing

    return {
      status: 'completed',
      confidence: 0.85,
      findings: ['Normal sinus rhythm', 'No significant arrhythmias'],
      recommendations: ['Continue monitoring']
    };
  }
}
```

Security Implementation

Authentication and Authorization

```
// JWT token management
class AuthService {
  generateTokens(user: User): TokenPair {
    const accessToken = jwt.sign(
      { userId: user.id, email: user.email },
      process.env.JWT_SECRET!,
      { expiresIn: '15m' }
    );

    const refreshToken = jwt.sign(
      { userId: user.id, tokenType: 'refresh' },
      process.env.REFRESH_SECRET!,
      { expiresIn: '30d' }
    );

    return { accessToken, refreshToken };
  }

  async validateToken(token: string): Promise<User | null> {
    try {
      const payload = jwt.verify(token, process.env.JWT_SECRET!) as JWTPayload;
      return await this.getUserById(payload.userId);
    } catch (error) {
      return null;
    }
  }
}
```

Data Encryption

```
// Health data encryption
class EncryptionService {
  private algorithm = 'aes-256-gcm';

  encryptHealthData(data: any): EncryptedData {
    const key = crypto.scryptSync(process.env.ENCRYPTION_KEY!, 'salt', 32);
    const iv = crypto.randomBytes(16);
    const cipher = crypto.createCipher(this.algorithm, key, iv);

    let encrypted = cipher.update(JSON.stringify(data), 'utf8', 'hex');
    encrypted += cipher.final('hex');

    const authTag = cipher.getAuthTag();

    return {
      encrypted,
      iv: iv.toString('hex'),
      authTag: authTag.toString('hex')
    };
  }

  decryptHealthData(encryptedData: EncryptedData): any {
    const key = crypto.scryptSync(process.env.ENCRYPTION_KEY!, 'salt', 32);
    const decipher = crypto.createDecipher(
      this.algorithm,
      key,
      Buffer.from(encryptedData.iv, 'hex')
    );

    decipher.setAuthTag(Buffer.from(encryptedData.authTag, 'hex'));

    let decrypted = decipher.update(encryptedData.encrypted, 'hex', 'utf8');
    decrypted += decipher.final('utf8');

    return JSON.parse(decrypted);
  }
}
```

User Experience Guidelines

Design Principles

Visual Design

- **Color Scheme**:
- Primary: Blue (#1976D2) for navigation and primary actions
- Secondary: Green (#4CAF50) for positive status indicators
- Warning: Orange (#FF9800) for attention-required states

- Error: Red (#F44336) for error states and alerts
- Background: Light gray (#F5F5F5) for main background
- Cards: White (#FFFFFF) with subtle shadows

Typography

- **Headers**: Bold, 18-24px for screen titles
- **Body Text**: Regular, 14-16px for content
- **Captions**: Light, 12-14px for timestamps and metadata
- **Font Family**: System fonts (San Francisco on iOS, Roboto on Android)

Spacing and Layout

- **Margins**: 16px standard margin for screen edges
- **Padding**: 12px for card content, 8px for compact elements
- **Grid System**: 8px base unit for consistent spacing
- **Touch Targets**: Minimum 44px for interactive elements

Accessibility Guidelines

Screen Reader Support

- **Semantic Labels**: All interactive elements have descriptive labels
- **Content Description**: Complex UI elements have detailed descriptions
- **Navigation Hints**: Clear indication of navigation structure
- **State Announcements**: Changes in app state are announced

Visual Accessibility

- **Color Contrast**: WCAG AA compliance (4.5:1 ratio minimum)
- **Text Scaling**: Support for dynamic type sizing
- **Focus Indicators**: Clear visual focus for keyboard navigation
- **Alternative Text**: All images have descriptive alt text

Motor Accessibility

- **Touch Target Size**: Minimum 44x44pt touch targets
- **Gesture Alternatives**: Alternative input methods for complex gestures
- **Timeout Extensions**: Configurable timeouts for timed interactions
- **Error Prevention**: Clear validation and confirmation dialogs

Performance Guidelines

Loading and Response Times

- **App Launch**: < 3 seconds to interactive state
- **Screen Transitions**: < 300ms animation duration
- **API Responses**: < 2 seconds for data loading
- **Real-time Updates**: < 100ms latency for ECG streaming

Memory and Battery Optimization

- **Memory Usage**: < 100MB baseline memory footprint
- **Battery Impact**: Optimized Bluetooth scanning and data processing
- **Background Processing**: Minimal background activity
- **Data Caching**: Intelligent caching to reduce network requests

Offline Functionality

- **Core Features**: Basic app functionality available offline
- **Data Sync**: Automatic sync when connection restored
- **Conflict Resolution**: Clear handling of data conflicts
- **Storage Management**: Automatic cleanup of old cached data

Error Handling and Recovery

Error States

- **Network Errors**: Clear messaging with retry options

- **Device Connection**: Step-by-step troubleshooting guidance
- **Data Validation**: Inline validation with helpful error messages
- **System Errors**: Graceful degradation with fallback options

Recovery Mechanisms

- **Automatic Retry**: Intelligent retry logic for transient failures
- **Manual Recovery**: Clear recovery actions for user-initiated fixes
- **Data Recovery**: Backup and restore capabilities for critical data
- **Support Integration**: Easy access to help and support resources

Conclusion

This comprehensive documentation provides a complete blueprint for reconstructing the SCLA (Skiin Connected Life App) mobile application. The analysis covers all aspects from user interface design and navigation flows to backend architecture and data models.

Key Implementation Priorities

1. **Real-time ECG Monitoring**: Core functionality requiring robust WebSocket implementation and optimized data visualization
2. **Device Integration**: Reliable Bluetooth connectivity with comprehensive error handling
3. **Health Data Management**: Secure, HIPAA-compliant storage and processing of sensitive health information
4. **User Experience**: Intuitive interface design with accessibility and performance optimization
5. **Clinical Integration**: Seamless workflow for healthcare providers and clinical studies

Next Steps for Implementation

1. **Technical Architecture Setup**: Establish development environment and core infrastructure

2. **Authentication System**: Implement secure user authentication with multiple sign-in methods
3. **Device Pairing Flow**: Develop Bluetooth connectivity and device management
4. **Core UI Components**: Build reusable components for consistent user experience
5. **Real-time Data Streaming**: Implement ECG data visualization and processing
6. **Health Data Entry**: Create symptom and blood pressure logging functionality
7. **Diary and Analytics**: Develop historical data viewing and analysis features
8. **Testing and Validation**: Comprehensive testing including clinical validation
9. **Deployment and Monitoring**: Production deployment with monitoring and analytics

This documentation serves as a complete reference for development teams to recreate the SCLA app with full functionality and clinical-grade reliability.