```mermaid componentDiagram title Pro Mood Tracker - Component Architecture

package "Frontend Layer" {  
 component [User Interface] as UI {  
 component [Navigation System]  
 component [Theme Provider]  
 component [Screen Components]  
 component [UI Components Library]  
 }  
  
 component [State Management] as State {  
 component [Redux Store]  
 component [Context Providers]  
 component [Action Creators]  
 component [Reducers]  
 }  
  
 component [Core Logic] as Logic {  
 component [Authentication Logic]  
 component [Mood Analysis]  
 component [Notification Management]  
 component [Data Sync Manager]  
 }  
}  
  
package "Services Layer" {  
 component [API Services] as API {  
 component [Auth Service]  
 component [User Service]  
 component [Mood Service]  
 component [Analytics Service]  
 component [Notification Service]  
 }  
  
 component [Utility Services] as Utils {  
 component [Local Storage]  
 component [Logger]  
 component [Error Handler]  
 component [Network Monitor]  
 component [Date Utilities]  
 }  
}  
  
package "Data Layer" {  
 component [Firebase SDK] as Firebase {  
 component [Firebase Auth]  
 component [Firestore]  
 component [Firebase Storage]  
 component [Cloud Functions]  
 component [Analytics]  
 }  
  
 component [Local Data] as Local {  
 component [SQLite/Realm]  
 component [AsyncStorage]  
 component [Secure Storage]  
 }  
}  
  
package "External Integrations" {  
 component [Third-Party APIs] as External {  
 component [Weather API]  
 component [Location Services]  
 component [Social Auth Providers]  
 component [Push Notification Service]  
 }  
}  
  
' Main connections between packages  
UI --> State : uses  
State --> Logic : implements  
Logic --> API : calls  
API --> Firebase : integrates  
API --> External : requests  
Logic --> Utils : utilizes  
API --> Local : persists  
UI --> Utils : uses  
  
' Specific connections  
[Authentication Logic] --> [Auth Service]  
[Mood Analysis] --> [Mood Service]  
[Notification Management] --> [Notification Service]  
[Data Sync Manager] --> [Network Monitor]  
[Auth Service] --> [Firebase Auth]  
[Mood Service] --> [Firestore]  
[Mood Service] --> [Firebase Storage]  
[Notification Service] --> [Push Notification Service]  
[Local Storage] --> [SQLite/Realm]  
[Local Storage] --> [AsyncStorage]  
[Auth Service] --> [Secure Storage]  
[Mood Service] --> [Weather API]  
[Mood Service] --> [Location Services]  
[Auth Service] --> [Social Auth Providers]

}

## Figure 4.15: Component Diagram - Pro Mood Tracker Architecture

This component diagram illustrates the architectural structure of the Pro Mood Tracker application, showing the primary components, their organization into layers, and their relationships.

### Key Architectural Layers:

#### 1. Frontend Layer

* **User Interface (UI)**: Contains all visual elements organized into screens, with a consistent navigation system and themeable components.
* **State Management**: Implements Redux and Context API to manage application state across components.
* **Core Logic**: Houses business logic for authentication, mood analysis, notifications, and data synchronization.

#### 2. Services Layer

* **API Services**: Abstracts the interaction with backend services, including authentication, user management, mood tracking, analytics, and notifications.
* **Utility Services**: Provides cross-cutting functionalities like local storage, logging, error handling, network monitoring, and date/time calculations.

#### 3. Data Layer

* **Firebase SDK**: Integrates with Firebase services for authentication, database storage, media storage, serverless functions, and user analytics.
* **Local Data**: Manages local data persistence using SQLite/Realm for structured data, AsyncStorage for key-value pairs, and Secure Storage for sensitive information.

#### 4. External Integrations

* **Third-Party APIs**: Connects with external services to enhance application functionality, including weather information, location services, social authentication providers, and push notification services.

### Key Architectural Patterns:

1. **Layered Architecture**: The system is organized into distinct layers with clear responsibilities, promoting separation of concerns and improved maintainability.
2. **Service-Oriented Design**: Functionality is encapsulated within service components that provide well-defined interfaces to the rest of the application.
3. **Offline-First Approach**: Local data storage components ensure the application functions seamlessly without continuous internet connectivity.
4. **Clean API Abstraction**: API services abstract the implementation details of both Firebase and third-party services, making it easier to modify or replace these components.
5. **Centralized State Management**: Redux store and Context providers offer a predictable state container that facilitates data flow throughout the application.
6. **UI Component Composition**: Screen components are built from smaller, reusable UI components, promoting consistency and reducing duplication.

This component architecture enables the Pro Mood Tracker application to scale effectively while maintaining clean separation between presentation, business logic, and data access concerns. The design supports both online and offline operation, ensures security through proper authentication flows, and provides a foundation for future feature expansion.