# PRO MOOD TRACKER

## A Comprehensive Mood Tracking Application

---

# Title Page

\*\*PRO MOOD TRACKER\*\*

A Comprehensive Mood Tracking and Emotional Well-being Application

Submitted in partial fulfillment of the requirements for the Project Report

---

# Certificate

This is to certify that the project work titled "Pro Mood Tracker: A Comprehensive Mood Tracking and Emotional Well-being Application" is a bonafide work carried out by [Student Name], under the guidance of [Guide Name], submitted in partial fulfillment of the requirements for the award of [Degree Name].

---

# Abstract

Pro Mood Tracker is a comprehensive mood tracking application designed to help users log, visualize, and analyze their moods over time. The application serves as a personal diary for emotional well-being, providing deep insights into mood patterns and offering suggestions for improvement.

The application features a user-friendly interface that allows users to record their moods using a color-coded system, along with additional context such as time of day, notes, and weather conditions. Advanced analytics provide users with valuable insights into their emotional patterns, helping them identify triggers and trends. The application also includes a rewards system to encourage consistent mood logging and features that promote self-reflection and emotional growth.

Developed using React with TypeScript and Material-UI, the application prioritizes user privacy, data security, and a personalized experience. Pro Mood Tracker aims to be a valuable tool for anyone looking to improve their emotional awareness and overall well-being.

\*\*Keywords:\*\* Mood Tracking, Emotional Well-being, Mental Health, Data Visualization, Analytics

---

# Acknowledgment

I would like to express my sincere gratitude to my project guide, [Guide Name], for the continuous support, patience, motivation, and immense knowledge. Their guidance helped me throughout the research and development of this project.

I would also like to thank the faculty members of the [Department Name] for their valuable insights and suggestions that have contributed to the improvement of this project.

My sincere thanks also go to my peers and friends who have provided moral support and encouragement throughout this journey.

Finally, I would like to thank my family for their unwavering support and understanding during the course of this project.

---

# Table of Content

\*\*Title\*\* \*\*Page No.\*\*

- Title Page I

- Certificate II

- Abstract III

- Acknowledgment IV

- Table of Content V

## 1. Introduction

1.1 Project Summary 1

1.2 Project Scope 2

1.3 Objective 5

## 2. System Requirement Study

2.1 Hardware and Software Characteristics 12

## 3. System Analysis

3.1 Study of Current System and Requirement of this System 22

3.2 Data Dictionary 25

3.3 Modules and Their Description of System 32

3.4 Database Design 35

## 4. System Design

4.1 Design Pseudocode or Algorithm for Method or Operation 42

4.2 Use Case Diagram 43

4.3 ER Diagram 53

4.4 DFD Diagram 63

## 5. Screenshots

5.1 Frontend/Client/End User Side Screenshots 72

5.2 Backend/Admin Side Screenshots 85

5.3 Other Screenshots 92

## 6. Conclusion And Future Work 98

\*\*REFERENCES\*\* 100

---

# LIST OF FIGURES

- Fig 1.1 Use case / Procedure Diagram 2

- Fig 1.2 Activity / Process Diagram 5

- Fig 2.1 Use case / Procedure Diagram 12

- Fig 2.2 Activity / Process Diagram 15

- Fig 2.3 Use case / Procedure Diagram 22

- Fig 2.4 Activity / Process Diagram 25

- Fig 3.1 Use case / Procedure Diagram 32

- Fig 4.1 Activity / Process Diagram 35

- Fig 4.2 Use case / Procedure Diagram 42

- Fig 4.3 Activity / Process Diagram 43

---

# LIST OF TABLES

- Table 1.1 Popular Methods / Techniques 2

- Table 1.2 User / Reading / Observation Table 5

- Table 2.1 Popular Methods / Techniques 12

- Table 2.2 User / Reading / Observation Table 15

- Table 2.3 Popular Methods / Techniques 22

- Table 2.4 User / Reading / Observation Table 25

- Table 3.1 Popular Methods / Techniques 32

- Table 4.1 User / Reading / Observation Table 35

- Table 4.2 Popular Methods / Techniques 42

- Table 4.3 User / Reading / Observation Table 43

---

# 1. Introduction

## 1.1 Project Summary

Pro Mood Tracker is a comprehensive web application designed to help users track, analyze, and understand their emotional well-being over time. The application employs a user-friendly interface built with React and TypeScript, allowing users to log their moods, visualize their emotional patterns, and gain valuable insights into factors affecting their mental health.

In today's fast-paced world, mental health awareness has gained significant importance. People are increasingly recognizing the value of emotional self-awareness and the impact of mood patterns on overall well-being. Pro Mood Tracker addresses this need by providing a digital platform for mood journaling with advanced analytical capabilities.

The application allows users to record their moods using a color-coded system ranging from "Very Bad" to "Very Good," along with additional context such as time of day, notes, and weather conditions. This data is then transformed into intuitive visualizations and statistics, helping users identify triggers, patterns, and trends in their emotional states.

Beyond simple tracking, Pro Mood Tracker incorporates features designed to promote positive behavioral changes:

1. \*\*Mood Logging\*\*: Simple and intuitive mood recording with contextual information

2. \*\*Visual Analytics\*\*: Multiple chart types for comprehensive mood pattern analysis

3. \*\*Personalized Insights\*\*: Data-driven observations about mood trends and correlations

4. \*\*Rewards System\*\*: Gamification elements to encourage consistent tracking

5. \*\*Theme Customization\*\*: Personalized visual experience to suit user preferences

6. \*\*Data Management\*\*: Privacy-focused tools for exporting and managing personal data

The application is built using modern web development technologies, with React and TypeScript forming the foundation, Material-UI providing the component library, and various chart libraries enabling sophisticated data visualization.

## 1.2 Project Scope

The scope of the Pro Mood Tracker project encompasses the design, development, and implementation of a comprehensive mood tracking application that allows users to monitor and analyze their emotional well-being over time. The application aims to provide both immediate and long-term value through mood logging, visualization, analytics, and personalized recommendations.

### Fig 1.1 Use case / Procedure Diagram

```

┌─────────────────────────────────────────────────────────┐

│ Pro Mood Tracker │

└───────────────────────────┬─────────────────────────────┘

│

┌─────────────────┴─────────────────┐

▼ ▼

┌─────────────────┐ ┌─────────────────┐

│ End User │ │ System Admin │

└───────┬─────────┘ └────────┬────────┘

│ │

▼ ▼

┌─────────────────┐ ┌─────────────────┐

│ Register/Login │ │ Manage Database │

└───────┬─────────┘ └────────┬────────┘

│ │

▼ ▼

┌─────────────────┐ ┌─────────────────┐

│ Log Moods │ │ Manage Settings │

└───────┬─────────┘ └────────┬────────┘

│ │

▼ ▼

┌─────────────────┐ ┌─────────────────┐

│ View Analytics │ │ Monitor Activity │

└───────┬─────────┘ └────────┬────────┘

│ │

▼ ▼

┌─────────────────┐ ┌─────────────────┐

│ Receive Insights│ │ Generate Reports│

└───────┬─────────┘ └────────┬────────┘

│ │

▼ ▼

┌─────────────────┐ ┌─────────────────┐

│Manage Preferences│ │ System Backup │

└─────────────────┘ └─────────────────┘

```

### Table 1.1 Popular Methods / Techniques

| Method/Technique | Description | Application in Pro Mood Tracker |

|------------------|-------------|--------------------------------|

| React Functional Components | Modern approach to building UI components with hooks | Used throughout the application for all UI components |

| Context API | State management solution for React applications | Used for mood data, user data, authentication, and theme management |

| Material-UI Components | Pre-built React components following Material Design | Forms the foundation of the UI with cards, buttons, inputs, etc. |

| Recharts Library | Composable charting library for React | Powers the analytics visualizations including line, bar, and pie charts |

| TypeScript Interfaces | Static type definitions for JavaScript | Ensures type safety across the application for data structures |

| Local Storage | Browser-based client-side storage | Persists user data and preferences between sessions |

| Responsive Design | UI adaptation for different screen sizes | Ensures consistent experience across desktop and mobile devices |

| JWT Authentication | Token-based user authentication | Secures user accounts and protects personal data |

### Project Inclusions:

1. \*\*User Authentication System\*\*:

- Secure user registration and login

- Password reset functionality

- User profile management

2. \*\*Mood Logging Functionality\*\*:

- Color-coded mood entry system

- Time period selection (morning, afternoon, evening, night, full-day)

- Notes and context recording

- Weather data integration

3. \*\*Visualization Tools\*\*:

- Interactive calendar with color-coded mood indicators

- Multiple chart types:

- Line graphs for mood trends

- Bar charts for mood distribution

- Radar/Spider charts for time-of-day analysis

- Correlation charts for weather impact

4. \*\*Analytics Features\*\*:

- Mood statistics and trends

- Day of week analysis

- Time of day patterns

- Weather correlation insights

- Mood predictions based on historical data

5. \*\*History Management\*\*:

- Searchable mood history

- Filter by mood, date, and time of day

- Detailed view of past entries

- Edit and delete functionality

6. \*\*Rewards System\*\*:

- Points for consistent logging

- Badges for achievements

- Streak tracking

- Progress visualization

7. \*\*Customization Options\*\*:

- Multiple theme options (Light, Dark, Ocean, Sunset, Forest, Pastel)

- User interface preferences

8. \*\*Data Management\*\*:

- Export options (JSON, CSV)

- Email functionality

- Data privacy controls

### Project Exclusions:

1. Full-scale backend implementation with database storage (using mock data and localStorage for demonstration)

2. Integration with third-party APIs (weather data is simulated)

3. Mobile native applications (focuses on web application only)

4. Social sharing features (privacy-focused approach)

5. Professional mental health analysis or medical recommendations

## 1.3 Objective

The primary objective of the Pro Mood Tracker application is to provide users with a comprehensive tool for tracking, analyzing, and improving their emotional well-being through data-driven insights and personalized recommendations.

### Fig 1.2 Activity / Process Diagram

```

┌─────────────────┐

│ Start Session │

└────────┬────────┘

│

▼

┌─────────────────┐

│ Authentication │

└────────┬────────┘

│

▼

┌─────────────────┐ ┌─────────────────┐

│ Dashboard View │────▶│ Log New Mood │

└────────┬────────┘ └────────┬────────┘

│ │

▼ │

┌─────────────────┐ │

│ View Calendar │◀─────────────┘

└────────┬────────┘

│

▼

┌─────────────────┐

│ View Analytics │

└────────┬────────┘

│

▼

┌─────────────────┐

│ Get Insights │

└────────┬────────┘

│

▼

┌─────────────────┐

│ Access Settings │

└────────┬────────┘

│

▼

┌─────────────────┐

│ End Session │

└─────────────────┘

```

### Table 1.2 User / Reading / Observation Table

| User Type | Primary Needs | Observation | Application Solution |

|-----------|---------------|-------------|----------------------|

| General Users | Track daily moods | Users want a simple way to record how they feel | Intuitive mood logging interface with emoji-based selection |

| Mental Health Conscious | Identify mood patterns | Users need to visualize trends over time | Multiple visualization types (calendar, charts, graphs) |

| Detail-Oriented | Record context with moods | Users want to add notes and details | Comprehensive note-taking and contextual factors (weather, activities) |

| Data-Driven | Analyze mood correlations | Users seek to understand what affects their mood | Advanced analytics showing correlations and insights |

| Consistency-Motivated | Stay engaged with tracking | Users benefit from incentives to maintain tracking | Rewards system with badges, streaks, and achievements |

| Privacy-Focused | Control personal data | Users concerned about sensitive information | Secure authentication and local data storage options |

| Personalization-Seeking | Customize experience | Users prefer personalized interfaces | Theme options and UI customization features |

### Specific Objectives:

1. \*\*Improve Emotional Self-Awareness\*\*:

- Enable users to recognize and name their emotions

- Provide a structured way to observe emotional patterns

- Promote reflection on factors affecting mood

2. \*\*Facilitate Pattern Recognition\*\*:

- Transform mood data into visual patterns

- Highlight correlations between activities and emotions

- Identify trends over different time periods (days, weeks, months)

3. \*\*Promote Positive Habit Formation\*\*:

- Encourage regular mood tracking through gamification

- Build consistency through streak tracking

- Reward progress with achievements and badges

4. \*\*Provide Actionable Insights\*\*:

- Generate personalized recommendations based on mood patterns

- Suggest activities that correlate with positive moods

- Offer strategies for managing identified negative triggers

5. \*\*Ensure Data Privacy and Control\*\*:

- Implement secure authentication

- Provide data export capabilities

- Ensure user ownership of personal data

6. \*\*Create an Engaging User Experience\*\*:

- Design an intuitive and aesthetic interface

- Offer customization options for personal preference

- Ensure accessibility across devices

7. \*\*Support Mental Health Awareness\*\*:

- Provide educational content about emotional well-being

- Normalize mood tracking as a wellness practice

- Offer resources for additional mental health support

---

# 2. System Requirement Study

## 2.1 Hardware and Software Characteristics

The Pro Mood Tracker application is designed as a modern web application that prioritizes accessibility, performance, and user experience. The system requirements encompass both the development environment needs and the end-user requirements for optimal usage.

### Fig 2.1 Use case / Procedure Diagram

```

┌─────────────────────────────────────────────────────┐

│ System Requirements │

└───────────────────────┬─────────────────────────────┘

│

┌─────────────────┴─────────────────┐

▼ ▼

┌──────────────┐ ┌──────────────┐

│ Development │ │ Deployment │

└──────┬───────┘ └──────┬───────┘

│ │

▼ ▼

┌──────────────┐ ┌──────────────┐

│ Hardware │ │ Hardware │

└──────┬───────┘ └──────┬───────┘

│ │

▼ ▼

┌──────────────┐ ┌──────────────┐

│ Software │ │ Software │

└──────┬───────┘ └──────┬───────┘

│ │

▼ ▼

┌──────────────┐ ┌──────────────┐

│ Dependencies│ │ Compatibility │

└──────────────┘ └──────────────┘

```

### Table 2.1 Popular Methods / Techniques

| Method/Technique | Description | Application in System Requirements |

|------------------|-------------|-----------------------------------|

| Progressive Web App (PWA) | Web apps that work offline with app-like features | Enables offline mood logging and improved mobile experience |

| Responsive Web Design | Design approach for different screen sizes | Ensures usability across desktop, tablet, and mobile devices |

| Local Storage API | Browser-based data persistence | Stores user data and preferences without server requirement |

| Lazy Loading | Technique to load resources only when needed | Improves initial load time and performance |

| Code Splitting | Breaking code into smaller chunks | Reduces main bundle size for faster loading |

| Compression | Reducing file sizes for transfer | Minimizes bandwidth usage and improves load times |

| Browser Compatibility Testing | Ensuring functionality across browsers | Guarantees consistent experience regardless of browser choice |

| Accessibility Standards | Design for users with disabilities | Ensures application is usable by all potential users |

### Development Environment Requirements

#### Hardware Requirements:

1. \*\*Computer System\*\*:

- Processor: Intel Core i5 (8th generation or higher) or equivalent AMD processor

- RAM: Minimum 8GB (16GB recommended)

- Storage: At least 256GB SSD (for development tools and codebase)

- Display: 1920 x 1080 resolution or higher

2. \*\*Development Tools\*\*:

- Code Editor/IDE: Visual Studio Code, WebStorm, or equivalent

- Browser: Latest versions of Chrome, Firefox, Safari for testing

- Version Control: Git client

- Terminal/Command Line Interface

#### Software Requirements:

1. \*\*Operating System\*\*:

- Windows 10/11, macOS 10.15 or later, or Linux distribution (Ubuntu 18.04 or newer)

2. \*\*Development Stack\*\*:

- Node.js (v14.x or higher)

- npm (v6.x or higher) or Yarn (v1.22.x or higher)

- Git (v2.x or higher)

3. \*\*Core Technologies\*\*:

- React (v18.2.0)

- TypeScript (v4.9.5)

- Material-UI (v6.4.7)

4. \*\*Key Libraries and Dependencies\*\*:

- React Router (v7.2.0) for navigation

- Recharts (v2.15.1) for data visualization

- Chart.js (v4.4.8) for additional charts

- date-fns (v2.30.0) for date manipulation

- UUID (v11.1.0) for unique identifiers

- React Context API for state management

- jsPDF (v3.0.0) for PDF generation

### Fig 2.2 Activity / Process Diagram

```

┌─────────────────┐

│ Development │

│ Environment │

└────────┬────────┘

│

▼

┌─────────────────┐

│ Code & Build │

└────────┬────────┘

│

▼

┌─────────────────┐

│ Testing │

└────────┬────────┘

│

▼

┌─────────────────┐

│ Deployment │

└────────┬────────┘

│

▼

┌─────────────────┐

│ User Access │

└────────┬────────┘

│

▼

┌─────────────────┐

│ User Feedback │

└────────┬────────┘

│

▼

┌─────────────────┐

│ Iteration │

└─────────────────┘

```

### End-User Requirements

#### Hardware Requirements:

1. \*\*Computer/Device\*\*:

- Desktop/Laptop: Any modern computer capable of running a supported web browser

- Mobile Devices: Any smartphone or tablet with a supported web browser

- Minimum Screen Resolution: 320px width (mobile), 768px width (desktop)

2. \*\*Input Devices\*\*:

- Mouse/Trackpad and Keyboard for desktop users

- Touchscreen for mobile/tablet users

3. \*\*Internet Connection\*\*:

- Any active internet connection for initial loading

- Application can function offline after initial load using PWA capabilities

#### Software Requirements:

1. \*\*Web Browsers\*\*:

- Google Chrome (v88 or higher)

- Mozilla Firefox (v87 or higher)

- Safari (v14 or higher)

- Microsoft Edge (Chromium-based, v88 or higher)

- Mobile browsers: Chrome for Android, Safari for iOS

2. \*\*Additional Requirements\*\*:

- JavaScript enabled

- Local Storage enabled

- Cookies enabled (for authentication)

### Table 2.2 User / Reading / Observation Table

| Requirement Type | Specification | Purpose | Impact on User Experience |

|------------------|---------------|---------|---------------------------|

| Display Resolution | Minimum 320px width | Mobile accessibility | Ensures usability on small screens |

| Processor Speed | Modern CPU (last 5 years) | Application performance | Smooth animations and transitions |

| Memory (RAM) | 2GB minimum | Data processing | Handles large mood history datasets |

| Storage | 50MB free space | PWA and cache storage | Enables offline functionality |

| Browser Support | Chrome, Firefox, Safari, Edge | Cross-platform accessibility | Consistent experience across browsers |

| Internet Speed | 1 Mbps minimum | Initial loading | Quick application startup |

| Touch Support | Touch events | Mobile usability | Natural interaction on touchscreens |

| Color Display | 16-bit color depth | UI visualization | Accurate representation of mood colors |

### System Architecture

Pro Mood Tracker employs a client-side architecture with the following components:

1. \*\*Presentation Layer\*\*:

- React components for UI rendering

- Material-UI for styling and component library

- CSS for custom styling and animations

2. \*\*Application Layer\*\*:

- React Context API for state management

- Custom hooks for business logic

- Service modules for data processing

3. \*\*Data Layer\*\*:

- Browser's Local Storage for data persistence

- In-memory state management

- Export/import functionality for data backup

### Fig 2.3 Use case / Procedure Diagram

```

┌───────────────────────────────────────────────────────┐

│ System Architecture │

└───────────────────────┬───────────────────────────────┘

│

┌─────────────────┴──────────────────┐

▼ ▼

┌──────────────┐ ┌──────────────┐

│ Presentation │ │ Application │

└──────┬───────┘ └──────┬───────┘

│ │

▼ ▼

┌──────────────┐ ┌──────────────┐

│ Components │ │ Context │

└──────┬───────┘ └──────┬───────┘

│ │

▼ ▼

┌──────────────┐ ┌──────────────┐

│ Pages │ │ Services │

└──────┬───────┘ └──────┬───────┘

│ │

▼ ▼

┌──────────────┐ ┌──────────────┐

│ UI Elements │ │ Data Storage │

└──────────────┘ └──────────────┘

```

### Performance Requirements

1. \*\*Load Time\*\*:

- Initial application load: < 3 seconds on broadband connections

- Subsequent page navigations: < 300ms

2. \*\*Responsiveness\*\*:

- UI interactions (button clicks, form inputs): < 100ms response time

- Data visualizations rendering: < 1 second for complex charts

3. \*\*Offline Capability\*\*:

- Application should function without internet connection after initial load

- Data should persist between sessions using local storage

4. \*\*Resource Usage\*\*:

- Memory usage: < 100MB in typical usage scenarios

- CPU usage: < 15% on average systems

### Table 2.3 Popular Methods / Techniques

| Method/Technique | Description | Application in Performance |

|------------------|-------------|----------------------------|

| Code Splitting | Dividing code into smaller chunks | Reduces initial load time |

| Memoization | Caching results of expensive function calls | Improves rendering performance |

| Virtual DOM | Virtual representation of UI | Efficient UI updates |

| React.memo | Preventing unnecessary re-renders | Performance optimization for components |

| useCallback/useMemo | Hooks for memoization | Prevents recreating functions and values |

| Tree Shaking | Eliminating dead code | Reduces bundle size |

| Image Optimization | Compressing and resizing images | Faster loading of visual assets |

| Lazy Loading | Loading components only when needed | Improves initial page load time |

### Security Requirements

1. \*\*Authentication\*\*:

- Secure user registration and login process

- Password hashing for stored credentials

- Session timeout after period of inactivity

2. \*\*Data Privacy\*\*:

- All personal data stored locally on user's device

- Optional encrypted export for data backups

- No transmission of mood data to external servers

3. \*\*Application Security\*\*:

- Protection against cross-site scripting (XSS)

- Regular security updates for dependencies

- Input validation for all user-provided data

### Fig 2.4 Activity / Process Diagram

```

┌─────────────────┐

│ User Access │

└────────┬────────┘

│

▼

┌─────────────────┐

│ Authentication │

└────────┬────────┘

│

▼

┌─────────────────┐ ┌─────────────────┐

│ Authorization │────▶│ Access Denied │

└────────┬────────┘ └─────────────────┘

│

▼

┌─────────────────┐

│ Data Access │

└────────┬────────┘

│

▼

┌─────────────────┐

│ Local Storage │

└────────┬────────┘

│

▼

┌─────────────────┐

│ Data Processing │

└────────┬────────┘

│

▼

┌─────────────────┐

│ UI Rendering │

└─────────────────┘

```

### Table 2.4 User / Reading / Observation Table

| Security Aspect | Implementation | User Benefit | Technical Approach |

|-----------------|----------------|--------------|-------------------|

| Authentication | Email/Password | Account protection | JWT token-based auth |

| Data Storage | Local storage | Privacy control | Browser's encrypted storage |

| Password Security | Hashing | Prevents plain-text exposure | bcrypt algorithm |

| Session Management | Timeout system | Protection on shared devices | Automatic logout after inactivity |

| Input Validation | Form validation | Prevents errors | Client-side validation rules |

| Data Backup | Export functionality | Data preservation | Encrypted JSON/CSV export |

| Privacy Controls | User settings | Customizable privacy | Granular permission settings |

| Cross-site Scripting | Prevention | Protection from attacks | React's built-in XSS protection |

### Maintenance Requirements

1. \*\*Code Maintainability\*\*:

- Well-documented codebase with comments

- Consistent coding style following React best practices

- Modular architecture for easy updates

2. \*\*Testing\*\*:

- Unit tests for key components and logic

- Integration tests for feature workflows

- Cross-browser compatibility testing

3. \*\*Dependency Management\*\*:

- Regular updates to npm packages

- Security vulnerability monitoring

- Backward compatibility considerations

4. \*\*Documentation\*\*:

- User documentation and help guides

- Developer documentation

- API documentation for future extensions

---

# 3. System Analysis

## 3.1 Study of Current System and Requirement of this System

The development of Pro Mood Tracker was inspired by an analysis of existing mood tracking applications and the identification of gaps in their functionality, usability, and analytical capabilities. This section provides an overview of the current state of mood tracking systems and outlines the specific requirements that guided the development of this application.

### Analysis of Existing Mood Tracking Systems

Current mood tracking applications can be broadly categorized into several types:

1. \*\*Basic Mood Journals\*\*:

- Simple interfaces for recording daily moods

- Limited visualization capabilities

- Minimal analytics or pattern recognition

- Examples: Daylio, Moodnotes

2. \*\*Health-Focused Trackers\*\*:

- Integrated with broader health metrics (sleep, exercise)

- Correlation between physical health and mood

- Often subscription-based with premium features

- Examples: Moodfit, Youper

3. \*\*Therapy-Oriented Applications\*\*:

- Designed as companions to professional therapy

- CBT (Cognitive Behavioral Therapy) integration

- Guided exercises and interventions

- Examples: MoodMission, Woebot

4. \*\*Social Mood Platforms\*\*:

- Sharing capabilities with communities

- Support networks and group features

- Privacy concerns for sensitive information

- Examples: Sanvello, TalkLife

### Limitations of Current Systems

Through market research and user feedback analysis, several limitations in existing systems were identified:

1. \*\*Limited Visualization Options\*\*:

- Most applications offer only basic charts or calendar views

- Insufficient tools for complex pattern recognition

- Lack of customizable visualization parameters

2. \*\*Restricted Analytical Depth\*\*:

- Few systems offer multi-factor correlation analysis

- Limited insights into mood triggers and patterns

- Minimal predictive capabilities

3. \*\*Inflexible Mood Recording\*\*:

- Predetermined mood scales with little customization

- Insufficient contextual information capture

- Limited time-of-day specificity

4. \*\*Privacy and Data Ownership Concerns\*\*:

- Many applications store sensitive data on remote servers

- Limited data export capabilities for user ownership

- Subscription models restricting access to personal data

5. \*\*Engagement and Consistency Issues\*\*:

- Limited mechanisms to encourage regular tracking

- Minimal gamification or reward systems

- High abandonment rates after initial usage

### Requirements for Pro Mood Tracker

Based on the analysis of existing systems and identified limitations, the following key requirements were established for Pro Mood Tracker:

1. \*\*Comprehensive Mood Recording\*\*:

- Flexible mood scale with contextual information

- Multiple time-of-day entries

- Detailed notes and associated factors

- Weather integration for environmental context

2. \*\*Advanced Visualization Tools\*\*:

- Multiple chart types for different analytical perspectives

- Interactive visualizations for deeper exploration

- Customizable parameters and time ranges

- Calendar view with intuitive color coding

3. \*\*Sophisticated Analytics\*\*:

- Multi-factor correlation analysis

- Time-based pattern recognition

- Predictive mood forecasting

- Personalized insights and recommendations

4. \*\*Privacy-Centric Approach\*\*:

- Local data storage by default

- Comprehensive export capabilities

- User control over all personal information

- No mandatory cloud storage or sharing

5. \*\*Engagement-Focused Features\*\*:

- Rewards system with points and badges

- Streak tracking for consistent usage

- Achievement system for milestone recognition

- Personalized recommendations for improvement

6. \*\*Customization Options\*\*:

- Multiple theme choices for interface

- Customizable mood categories and scales

- User-defined tracking parameters

- Flexible reporting options

### Fig 3.1 Use case / Procedure Diagram

```

┌─────────────────────────────────────────────────────────┐

│ Pro Mood Tracker System │

└───────────────────────────┬─────────────────────────────┘

│

┌───────────────────────────┼───────────────────────────┐

│ │ │

▼ ▼ ▼

┌─────────────────┐ ┌─────────────────┐ ┌─────────────────┐

│ Mood Logging │ │ Visualization │ │ Analytics │

└───────┬─────────┘ └───────┬─────────┘ └───────┬─────────┘

│ │ │

▼ ▼ ▼

┌─────────────────┐ ┌─────────────────┐ ┌─────────────────┐

│Record Daily Mood│ │ View Calendar │ │ Track Patterns │

└───────┬─────────┘ └───────┬─────────┘ └───────┬─────────┘

│ │ │

▼ ▼ ▼

┌─────────────────┐ ┌─────────────────┐ ┌─────────────────┐

│ Add Context │ │ Explore Charts │ │ Get Insights │

└───────┬─────────┘ └───────┬─────────┘ └───────┬─────────┘

│ │ │

▼ ▼ ▼

┌─────────────────┐ ┌─────────────────┐ ┌─────────────────┐

│ Save Entry │ │ Export Reports │ │ View Predictions│

└─────────────────┘ └─────────────────┘ └─────────────────┘

```

### Table 3.1 Popular Methods / Techniques

| Method/Technique | Description | Application in Pro Mood Tracker |

|------------------|-------------|--------------------------------|

| User Persona Development | Creating fictional characters representing user types | Identified key user needs and preferences for different user types |

| Competitive Analysis | Evaluating similar products in the market | Identified gaps and opportunities in existing mood tracking applications |

| Feature Prioritization | Ranking features by importance and feasibility | Determined core vs. nice-to-have features for MVP and future releases |

| Usability Testing | Evaluating ease of use through user testing | Refined UI/UX based on actual user interaction and feedback |

| Heuristic Evaluation | Assessment against established usability principles | Ensured interface adheres to recognized UX best practices |

| Agile Development | Iterative approach with regular feedback | Allowed for continuous refinement and feature adjustments |

| Responsive Design | Adapting UI for different screen sizes | Ensured consistent experience across desktop and mobile devices |

| Accessibility Audit | Evaluating against WCAG guidelines | Made application usable by people with diverse abilities |

## 3.2 Data Dictionary

The data dictionary for Pro Mood Tracker defines the structure, relationships, and constraints of all data elements within the application. This comprehensive reference ensures consistency in data handling across the system and provides a foundation for future development and maintenance.

### Core Data Types

#### MoodType

```typescript

type MoodType = 'Very Bad' | 'Bad' | 'Okay' | 'Good' | 'Very Good';

```

Description: Represents the five possible mood states that users can select when logging their mood.

| Value | Numeric Equivalent | Color Representation | Description |

|-------|-------------------|----------------------|-------------|

| Very Bad | 0 | #E53935 (Red) | Extremely negative emotional state |

| Bad | 1 | #FB8C00 (Orange) | Negative emotional state |

| Okay | 2 | #FDD835 (Yellow) | Neutral emotional state |

| Good | 3 | #43A047 (Light Green) | Positive emotional state |

| Very Good | 4 | #1E88E5 (Blue) | Extremely positive emotional state |

#### TimeOfDay

```typescript

type TimeOfDay = 'morning' | 'afternoon' | 'evening' | 'night' | 'full-day';

```

Description: Represents the time period for which a mood is being recorded.

| Value | Time Range | Icon | Description |

|-------|------------|------|-------------|

| morning | 5:00 AM - 11:59 AM | WbSunnyIcon | Early part of the day, typically after waking |

| afternoon | 12:00 PM - 4:59 PM | Brightness4Icon | Middle part of the day |

| evening | 5:00 PM - 8:59 PM | WbTwilightIcon | Later part of the day, typically after work |

| night | 9:00 PM - 4:59 AM | NightsStayIcon | Nighttime hours, typically before sleep |

| full-day | All day | CalendarTodayIcon | Overall mood for the entire day |

### Primary Data Structures

#### MoodEntry

```typescript

interface MoodEntry {

id: string;

date: string; // ISO string format

timeOfDay: TimeOfDay;

mood: MoodType;

notes: string;

customCategory?: string;

time?: string; // Time in HH:MM format

journal?: string; // Journal entry text

activities?: string[]; // List of activities

weather?: {

temperature?: number;

condition?: string;

humidity?: number;

};

}

```

Description: The core data structure that represents a single mood logging event by the user.

| Field | Type | Required | Description |

|-------|------|----------|-------------|

| id | string | Yes | Unique identifier (UUID) |

| date | string | Yes | ISO formatted date string (e.g., "2023-04-15T10:30:00Z") |

| timeOfDay | TimeOfDay | Yes | The period of day for which mood is recorded |

| mood | MoodType | Yes | The selected mood value |

| notes | string | Yes | User's textual notes about their mood |

| customCategory | string | No | Optional user-defined mood category |

| time | string | No | Specific time in HH:MM format |

| journal | string | No | Extended journal entry text |

| activities | string[] | No | Array of activities associated with the mood |

| weather.temperature | number | No | Temperature in degrees Celsius |

| weather.condition | string | No | Weather condition (e.g., "Sunny", "Rainy") |

| weather.humidity | number | No | Humidity percentage |

#### User

```typescript

interface User {

id: string;

username: string;

email: string;

createdAt: string;

points: number; // For rewards system

badges: Badge[];

settings: UserSettings;

}

```

Description: Represents a user account in the application.

| Field | Type | Required | Description |

|-------|------|----------|-------------|

| id | string | Yes | Unique identifier (UUID) |

| username | string | Yes | User's chosen display name |

| email | string | Yes | User's email address for authentication |

| createdAt | string | Yes | ISO timestamp of account creation |

| points | number | Yes | Accumulated points from activities |

| badges | Badge[] | Yes | Array of earned achievement badges |

| settings | UserSettings | Yes | User preference settings |

#### MoodStats

```typescript

interface MoodStats {

averageMoodByDay: { [key: string]: number }; // 0-4 for Very Bad to Very Good

averageMoodByTimeOfDay: { [key in TimeOfDay]: number };

moodCountByType: { [key in MoodType]: number };

mostFrequentMood: MoodType;

moodTrend: 'improving' | 'declining' | 'stable';

moodCorrelations: {

weather?: {

temperature?: number; // correlation coefficient

condition?: { [key: string]: number }; // sunny, rainy, etc. with avg mood

};

dayOfWeek?: { [key: string]: number }; // day of week with avg mood

};

}

```

Description: Contains aggregated statistics and analysis of a user's mood entries.

| Field | Type | Description |

|-------|------|-------------|

| averageMoodByDay | Object | Maps day of week (0-6) to average mood value (0-4) |

| averageMoodByTimeOfDay | Object | Maps time of day to average mood value |

| moodCountByType | Object | Count of entries for each mood type |

| mostFrequentMood | MoodType | The most commonly recorded mood |

| moodTrend | string | Overall trend of moods over recent entries |

| moodCorrelations.weather.temperature | number | Correlation coefficient between temperature and mood |

| moodCorrelations.weather.condition | Object | Maps weather conditions to average mood values |

| moodCorrelations.dayOfWeek | Object | Maps days of week to average mood values |

### Supporting Data Structures

#### Badge

```typescript

interface Badge {

id: string;

name: string;

description: string;

imageUrl: string;

earnedAt: string;

}

```

Description: Represents an achievement badge earned by the user.

#### UserSettings

```typescript

interface UserSettings {

theme: ThemeType;

enableWeatherTracking: boolean;

enablePredictions: boolean;

location?: {

city: string;

country: string;

lat: number;

lon: number;

};

}

```

Description: User-configurable application settings.

#### MoodPrediction

```typescript

interface MoodPrediction {

date: string;

predictedMood: MoodType;

confidence: number; // 0-1

}

```

Description: Represents a predicted future mood based on historical patterns.

#### MoodSuggestion

```typescript

interface MoodSuggestion {

forMood: MoodType;

suggestion: string;

}

```

Description: Personalized suggestion provided to the user based on their mood.

## 3.3 Modules and Their Description of System

Pro Mood Tracker is organized into distinct functional modules, each responsible for specific aspects of the application's functionality. This modular architecture promotes code organization, reusability, and maintainability.

### 1. Authentication Module

\*\*Purpose\*\*: Manages user authentication, registration, and session handling.

\*\*Components\*\*:

- Login component

- Registration component

- Password reset functionality

- Authentication context provider

- Protected route implementation

\*\*Functionality\*\*:

- User registration with email and password

- Secure login with credential validation

- Password reset via email

- Session persistence using JWT

- Route protection for authenticated users

- Profile management

\*\*Key Files\*\*:

- `src/components/auth/Login.tsx`

- `src/components/auth/Register.tsx`

- `src/context/AuthContext.tsx`

### 2. Mood Logging Module

\*\*Purpose\*\*: Enables users to record their moods with contextual information.

\*\*Components\*\*:

- Mood selection interface

- Time of day selector

- Date picker

- Notes and context recorder

- Weather integration

\*\*Functionality\*\*:

- Intuitive mood selection with visual feedback

- Multiple time period options

- Detailed note-taking capabilities

- Activity tagging

- Automatic weather data association

- Entry saving and updating

\*\*Key Files\*\*:

- `src/components/mood/MoodLogger.tsx`

- `src/context/MoodContext.tsx` (partial)

### 3. Visualization Module

\*\*Purpose\*\*: Provides visual representations of mood data across different time periods and formats.

\*\*Components\*\*:

- Calendar view with color-coding

- Timeline charts

- Distribution charts

- Correlation visualizations

- Customizable view options

\*\*Functionality\*\*:

- Interactive calendar with mood indicators

- Day, week, month, and year views

- Multiple chart types (line, bar, pie, radar)

- Drill-down capabilities for detailed analysis

- Custom date range selection

- Visual filtering options

\*\*Key Files\*\*:

- `src/components/mood/MoodCalendar.tsx`

- `src/components/charts/MoodAnalytics.tsx`

### 4. Analytics Module

\*\*Purpose\*\*: Processes mood data to provide insights, correlations, and predictions.

\*\*Components\*\*:

- Statistical analysis tools

- Pattern recognition algorithms

- Predictive modeling

- Insight generation

- Correlation analysis

\*\*Functionality\*\*:

- Calculation of mood trends and patterns

- Identification of mood correlations with factors

- Prediction of future moods based on patterns

- Generation of personalized insights

- Recommendation engine for mood improvement

\*\*Key Files\*\*:

- `src/components/charts/MoodAnalytics.tsx`

- `src/components/charts/PredictiveTrendAnalysis.tsx`

- `src/components/charts/CorrelationAnalysis.tsx`

- `src/services/moodService.ts`

### 5. History Management Module

\*\*Purpose\*\*: Allows users to access, search, and modify their historical mood entries.

\*\*Components\*\*:

- Searchable history list

- Filtering interface

- Entry detail view

- Edit and delete capabilities

\*\*Functionality\*\*:

- Chronological display of all mood entries

- Search by keyword, date, or mood

- Filtering by multiple parameters

- Detailed view of individual entries

- Modification and deletion of past entries

- Export capabilities for selected entries

\*\*Key Files\*\*:

- `src/components/history/MoodHistory.tsx`

### 6. Rewards and Gamification Module

\*\*Purpose\*\*: Encourages consistent mood tracking through gamification elements.

\*\*Components\*\*:

- Points system

- Badge and achievement display

- Streak tracking

- Progress visualization

\*\*Functionality\*\*:

- Award points for regular mood logging

- Track consecutive days streaks

- Issue badges for achieving milestones

- Display progress toward goals

- Provide motivational messages

\*\*Key Files\*\*:

- `src/components/rewards/RewardsSystem.tsx`

### 7. Customization Module

\*\*Purpose\*\*: Allows users to personalize their experience through various settings and preferences.

\*\*Components\*\*:

- Theme selector

- User preference settings

- Interface customization options

\*\*Functionality\*\*:

- Multiple visual themes (Light, Dark, Ocean, etc.)

- Custom mood category creation

- Interface layout preferences

- Notification settings

- Data privacy controls

\*\*Key Files\*\*:

- `src/components/themes/ThemePage.tsx`

- `src/themes/ThemeSwitcher.tsx`

- `src/context/ThemeContext.tsx`

### 8. Data Management Module

\*\*Purpose\*\*: Handles data persistence, export, import, and privacy controls.

\*\*Components\*\*:

- Local storage service

- Export functionality

- Import tools

- Data backup and restore

\*\*Functionality\*\*:

- Automatic saving to local storage

- Export to JSON and CSV formats

- Email data export

- Data backup and restore

- Privacy controls and data purging options

\*\*Key Files\*\*:

- Various service files

- Export/import utility functions

## 3.4 Database Design

Pro Mood Tracker utilizes a client-side data storage approach, primarily leveraging browser's Local Storage for persistence. While this isn't a traditional database in the server-based sense, the data structure and relationships are designed with similar considerations for integrity, efficiency, and scalability.

### Data Storage Approach

The application employs a hybrid storage strategy:

1. \*\*Local Storage\*\*: Primary persistence mechanism for all user data

- Mood entries

- User preferences

- Authentication tokens

- Application state

2. \*\*In-Memory State\*\*: Managed through React Context API

- Current user session

- Active mood entries

- Calculated statistics

- UI state

3. \*\*Export/Import\*\*: Optional JSON/CSV export for backup

- Full data export functionality

- Selective period export

- Import capabilities for restoration

### Entity Relationship Diagram

The following diagram represents the logical relationships between the main data entities in the application:

```

┌─────────────┐ ┌─────────────┐

│ User │ │ Settings │

├─────────────┤ ├─────────────┤

│ id │───1───┤ userId │

│ username │ │ theme │

│ email │ │ preferences │

│ createdAt │ └─────────────┘

│ points │

└─────────────┘

│

│

│1

│

▼n

┌─────────────┐ ┌─────────────┐

│ MoodEntry │ │ Weather │

├─────────────┤ ├─────────────┤

│ id │───1───┤ entryId │

│ userId │ │ temperature │

│ date │ │ condition │

│ timeOfDay │ │ humidity │

│ mood │ └─────────────┘

│ notes │

└─────────────┘

│

│1

│

▼n

┌─────────────┐ ┌─────────────┐

│ Activity │ │ Badge │

├─────────────┤ ├─────────────┤

│ id │ │ id │

│ entryId │ │ userId │

│ name │ │ name │

└─────────────┘ │ earnedAt │

└─────────────┘

```

### Data Storage Structure

#### Local Storage Keys

| Key | Purpose | Structure |

|-----|---------|-----------|

| `moodEntries` | Stores all mood entries | JSON array of MoodEntry objects |

| `user` | Current user information | User object |

| `authToken` | Authentication token | JWT string |

| `userSettings` | User preferences | UserSettings object |

| `achievements` | Earned badges and rewards | Array of Badge objects |

#### Sample Local Storage Implementation

```javascript

// Add a new mood entry

const addMoodEntry = (entry) => {

// Get existing entries

const existingEntries = JSON.parse(localStorage.getItem('moodEntries') || '[]');

// Add new entry with unique ID

const newEntry = {

...entry,

id: uuidv4(),

userId: currentUser.id

};

// Save updated entries

localStorage.setItem('moodEntries', JSON.stringify([...existingEntries, newEntry]));

// Update in-memory state

setMoodEntries([...existingEntries, newEntry]);

// Recalculate statistics

calculateAndUpdateStats();

};

```

### Data Access Patterns

The application follows consistent patterns for data access:

1. \*\*Initialization\*\*:

- Load data from Local Storage on application start

- Initialize React Context providers with loaded data

- Handle missing data with sensible defaults

2. \*\*Read Operations\*\*:

- Access data primarily through Context providers

- Filter and transform data as needed for specific views

- Cache complex calculations for performance

3. \*\*Write Operations\*\*:

- Update both Context state and Local Storage

- Maintain data integrity with validation

- Handle storage limitations gracefully

4. \*\*Export/Import\*\*:

- Provide formats suitable for external storage (JSON/CSV)

- Validate imported data before integration

- Manage version compatibility

### Data Migration and Versioning

To handle potential changes in data structure over time:

1. \*\*Version Tagging\*\*:

- All stored data includes a version identifier

- Enables smooth migration between versions

2. \*\*Migration Functions\*\*:

- Transform data between versions when necessary

- Run automatically during application updates

3. \*\*Backward Compatibility\*\*:

- Maintain support for older data formats

- Graceful degradation for missing fields

### Security Considerations

Despite being client-side storage, several security measures are implemented:

1. \*\*Sensitive Data Handling\*\*:

- Authentication tokens stored with expiration

- No storage of plain-text passwords

- Option for PIN protection of the application

2. \*\*Data Integrity\*\*:

- Validation before storage and after retrieval

- Checksums for exported data

- Recovery mechanisms for corrupted data

3. \*\*Privacy Controls\*\*:

- User options to purge sensitive data

- Clear distinction between essential and optional data

- Transparency about all stored information

---

# 4. System Design

## 4.1 Design Pseudocode or Algorithm for Method or Operation

The Pro Mood Tracker application implements several key algorithms for data processing, analysis, and visualization. This section presents the pseudocode for some of the core algorithms that power the application's functionality.

### Mood Entry Addition Algorithm

This algorithm handles the addition of new mood entries to the system, ensuring proper data validation, ID assignment, and statistics recalculation.

```

FUNCTION addMoodEntry(entry):

// Input validation

IF !isValidMood(entry.mood) OR !isValidTimeOfDay(entry.timeOfDay) OR !isValidDate(entry.date) THEN

THROW ValidationError("Invalid mood entry data")

END IF

// Generate unique ID

entry.id = generateUUID()

// Add timestamp if not provided

IF entry.date IS EMPTY THEN

entry.date = getCurrentISOTimestamp()

END IF

// Add weather data if enabled in settings

IF settings.enableWeatherTracking AND entry.weather IS EMPTY THEN

entry.weather = fetchWeatherData(entry.date, userLocation)

END IF

// Retrieve existing entries

existingEntries = getStoredMoodEntries()

// Add new entry

allEntries = existingEntries + entry

// Save updated entries

saveMoodEntries(allEntries)

// Recalculate statistics

updateMoodStatistics(allEntries)

// Update UI with new entry

notifyUIOfChanges()

RETURN entry

END FUNCTION

```

### Mood Statistics Calculation Algorithm

This algorithm processes all mood entries to generate statistical insights, correlations, and trends.

```

FUNCTION calculateMoodStats(moodEntries):

// Initialize statistics object

stats = {

averageMoodByDay: {},

averageMoodByTimeOfDay: {},

moodCountByType: {},

mostFrequentMood: null,

moodTrend: 'stable',

moodCorrelations: {

weather: { temperature: 0, condition: {} },

dayOfWeek: {}

}

}

// Count entries by mood type

FOR EACH entry IN moodEntries:

stats.moodCountByType[entry.mood] += 1

END FOR

// Determine most frequent mood

stats.mostFrequentMood = findKeyWithMaxValue(stats.moodCountByType)

// Calculate average mood by day of week

FOR dayIndex FROM 0 TO 6:

entriesForDay = filterEntriesByDayOfWeek(moodEntries, dayIndex)

stats.averageMoodByDay[dayIndex] = calculateAverageMoodValue(entriesForDay)

END FOR

// Calculate average mood by time of day

FOR EACH timeOfDay IN ['morning', 'afternoon', 'evening', 'night', 'full-day']:

entriesForTimeOfDay = filterEntriesByTimeOfDay(moodEntries, timeOfDay)

stats.averageMoodByTimeOfDay[timeOfDay] = calculateAverageMoodValue(entriesForTimeOfDay)

END FOR

// Calculate mood trend

recentEntries = getEntriesFromLastNDays(moodEntries, 30)

stats.moodTrend = analyzeMoodTrend(recentEntries)

// Calculate correlations with weather

IF hasWeatherData(moodEntries) THEN

stats.moodCorrelations.weather = calculateWeatherCorrelations(moodEntries)

END IF

// Calculate correlations with day of week

stats.moodCorrelations.dayOfWeek = calculateDayOfWeekCorrelations(moodEntries)

RETURN stats

END FUNCTION

```

### Mood Prediction Algorithm

This algorithm uses historical mood data to predict future moods based on patterns and correlations.

```

FUNCTION predictMood(moodEntries, targetDate):

// Extract features from target date

dayOfWeek = getDayOfWeek(targetDate)

month = getMonth(targetDate)

season = getSeason(targetDate, userLocation)

// Get similar historical dates

similarDates = findSimilarDates(moodEntries, dayOfWeek, month, season)

IF similarDates.length == 0 THEN

// Not enough data for prediction

RETURN {

date: targetDate.toISOString(),

predictedMood: getMostFrequentMood(moodEntries),

confidence: 0.3

}

END IF

// Count mood occurrences on similar dates

moodCounts = countMoodOccurrences(similarDates)

// Find most likely mood

likelyMood = findMoodWithHighestCount(moodCounts)

// Calculate confidence (0-1)

totalSimilarDates = similarDates.length

occurrencesOfLikelyMood = moodCounts[likelyMood]

confidence = occurrencesOfLikelyMood / totalSimilarDates

// Adjust confidence based on amount of data

IF totalSimilarDates < 5 THEN

confidence = confidence \* 0.7

END IF

RETURN {

date: targetDate.toISOString(),

predictedMood: likelyMood,

confidence: confidence

}

END FUNCTION

```

### Fig 4.1 Activity / Process Diagram

The following diagram illustrates the mood logging and analysis process:

```

┌─────────────────┐

│ User Input │

└────────┬────────┘

│

▼

┌─────────────────┐

│ Validation │

└────────┬────────┘

│

▼

┌─────────────────┐ ┌─────────────────┐

│ Data Storage │────▶│ Local Storage │

└────────┬────────┘ └─────────────────┘

│

▼

┌─────────────────┐

│ Processing │

└────────┬────────┘

│

▼

┌─────────────────┐ ┌─────────────────┐

│ Analytics │────▶│ Stats Calculation│

└────────┬────────┘ └─────────────────┘

│

▼

┌─────────────────┐

│ Visualization │

└────────┬────────┘

│

▼

┌─────────────────┐

│ User Feedback │

└─────────────────┘

```

### Table 4.1 User / Reading / Observation Table

| Process Stage | Technical Approach | User Experience Impact | Performance Consideration |

|---------------|-------------------|------------------------|---------------------------|

| Validation | Client-side validation with TypeScript | Immediate feedback on errors | Prevents invalid data in storage |

| Data Storage | Browser's Local Storage API | Data persistence between sessions | Limited storage capacity (5-10MB) |

| Processing | In-memory data transformation | Responsive interface during operations | CPU usage optimization needed for large datasets |

| Analytics | Statistical algorithms with memoization | Insightful data presentation | Computation caching for expensive operations |

| Visualization | Recharts and Chart.js rendering | Interactive and intuitive data exploration | Render optimization for smooth transitions |

| User Feedback | Material-UI notification system | Clear communication of system state | Minimal disruption to user workflow |

## 4.2 Use Case Diagram

The following use case diagram illustrates the primary interactions between users and the Pro Mood Tracker system:

### Fig 4.2 Use case / Procedure Diagram

```

┌─────────────────────────────────────────────────────┐

│ Pro Mood Tracker │

└─────────────────────────────────────────────────────┘

│

│

▼

┌─────────────────────────────────────────────────────┐

│ Mood Tracker App │

└───────────────────────┬─────────────────────────────┘

│

┌───────────────────┼────────────────────┐

│ │ │

▼ ▼ ▼

┌───────────┐ ┌───────────┐ ┌───────────┐

│ Log Mood │ │View Stats │ │ Manage │

│ │ │ │ │ Account │

└─────┬─────┘ └─────┬─────┘ └─────┬─────┘

│ │ │

▼ ▼ ▼

┌───────────┐ ┌───────────┐ ┌───────────┐

│ Regular │ │ Data │ │ Admin │

│ User │ │ Analyst │ │ │

└───────────┘ └───────────┘ └───────────┘

```

### Detailed Use Cases

#### Use Case 1: User Registration and Login

\*\*Actor\*\*: New/Existing User

\*\*Description\*\*:

The user registers for a new account or logs into an existing account to access the mood tracking features.

\*\*Preconditions\*\*:

- The user has internet access

- The application is loaded in the browser

\*\*Main Flow\*\*:

1. User selects "Register" or "Login" option

2. System displays the appropriate form

3. User enters required information

4. System validates the information

5. System creates new account or authenticates existing account

6. System redirects to the dashboard

\*\*Alternative Flows\*\*:

- Registration validation fails: System displays error messages

- Login credentials incorrect: System displays error message

- Password forgotten: System provides password reset option

\*\*Postconditions\*\*:

- User is authenticated and can access mood tracking features

- Authentication token is stored in local storage

#### Use Case 2: Logging a Mood Entry

\*\*Actor\*\*: Authenticated User

\*\*Description\*\*:

The user records their current emotional state with additional context.

\*\*Preconditions\*\*:

- User is authenticated

- User has accessed the mood logging interface

\*\*Main Flow\*\*:

1. User selects date for the entry

2. User selects time of day

3. User chooses mood from the available options

4. User adds optional notes or context

5. User submits the entry

6. System validates and stores the entry

7. System updates statistics and visualizations

\*\*Alternative Flows\*\*:

- Validation fails: System displays error messages

- Multiple entries for same time period: System asks for confirmation to overwrite

\*\*Postconditions\*\*:

- New mood entry is saved

- Dashboard and visualizations are updated with new data

### Table 4.2 Popular Methods / Techniques

| Method/Technique | Description | Application in Use Cases |

|------------------|-------------|--------------------------|

| JWT Authentication | Token-based authentication system | Secures user identity across sessions |

| Form Validation | Client-side data validation techniques | Ensures data integrity during mood logging |

| State Management | React Context API for application state | Maintains consistent UI state across components |

| Responsive Design | UI adaptability for different screen sizes | Ensures usability on mobile and desktop devices |

| Real-time Feedback | Immediate system response to user actions | Provides confirmation of mood logging success |

| Data Visualization | Chart rendering and interactive displays | Presents mood data in comprehensible formats |

| Local Storage | Browser-based data persistence | Maintains user data between sessions |

| Error Handling | Graceful management of exceptions | Prevents application crashes during use |

## 4.3 ER Diagram

The following Entity-Relationship diagram illustrates the data model used in Pro Mood Tracker:

### Fig 4.3 ER Diagram

```

┌─────────────┐ ┌─────────────┐

│ User │ │ Settings │

├─────────────┤ ├─────────────┤

│ id │───1───┤ userId │

│ username │ │ theme │

│ email │ │ preferences │

│ createdAt │ └─────────────┘

│ points │

└─────────────┘

│

│

│1

│

▼n

┌─────────────┐ ┌─────────────┐

│ MoodEntry │ │ Weather │

├─────────────┤ ├─────────────┤

│ id │───1───┤ entryId │

│ userId │ │ temperature │

│ date │ │ condition │

│ timeOfDay │ │ humidity │

│ mood │ └─────────────┘

│ notes │

└─────────────┘

│

│1

│

▼n

┌─────────────┐ ┌─────────────┐

│ Activity │ │ Badge │

├─────────────┤ ├─────────────┤

│ id │ │ id │

│ entryId │ │ userId │

│ name │ │ name │

└─────────────┘ │ earnedAt │

└─────────────┘

```

### Entity Descriptions

#### User Entity

- \*\*id\*\*: Unique identifier (UUID)

- \*\*username\*\*: User's chosen display name

- \*\*email\*\*: Email address for authentication

- \*\*createdAt\*\*: Timestamp of account creation

- \*\*points\*\*: Points accumulated in the rewards system

#### MoodEntry Entity

- \*\*id\*\*: Unique identifier (UUID)

- \*\*userId\*\*: Foreign key to User

- \*\*date\*\*: ISO timestamp of the entry

- \*\*timeOfDay\*\*: Period of the day (morning, afternoon, evening, night, full-day)

- \*\*mood\*\*: Emotional state (Very Bad, Bad, Okay, Good, Very Good)

- \*\*notes\*\*: Textual context for the mood

#### Weather Entity

- \*\*entryId\*\*: Foreign key to MoodEntry

- \*\*temperature\*\*: Temperature in degrees Celsius

- \*\*condition\*\*: Weather condition (e.g., Sunny, Rainy)

- \*\*humidity\*\*: Humidity percentage

#### Activity Entity

- \*\*id\*\*: Unique identifier (UUID)

- \*\*entryId\*\*: Foreign key to MoodEntry

- \*\*name\*\*: Activity description

#### Badge Entity

- \*\*id\*\*: Unique identifier (UUID)

- \*\*userId\*\*: Foreign key to User

- \*\*name\*\*: Badge name

- \*\*earnedAt\*\*: Timestamp when badge was earned

#### Settings Entity

- \*\*userId\*\*: Foreign key to User

- \*\*theme\*\*: Selected UI theme

- \*\*preferences\*\*: JSON-encoded user preferences

### Table 4.3 User / Reading / Observation Table

| Entity | Relationships | Cardinality | Key Attributes |

|--------|--------------|-------------|----------------|

| User | Has many MoodEntries | 1:N | id (Primary Key) |

| User | Has one Settings | 1:1 | id (Primary Key) |

| User | Has many Badges | 1:N | id (Primary Key) |

| MoodEntry | Belongs to one User | N:1 | id (Primary Key), userId (Foreign Key) |

| MoodEntry | Has one Weather | 1:1 | id (Primary and Foreign Key) |

| MoodEntry | Has many Activities | 1:N | id (Primary Key) |

| Weather | Belongs to one MoodEntry | N:1 | entryId (Primary and Foreign Key) |

| Activity | Belongs to one MoodEntry | N:1 | id (Primary Key), entryId (Foreign Key) |

| Badge | Belongs to one User | N:1 | id (Primary Key), userId (Foreign Key) |

| Settings | Belongs to one User | 1:1 | userId (Primary and Foreign Key) |

## 4.4 DFD Diagram

The following Data Flow Diagram illustrates how data moves through the Pro Mood Tracker system:

```

┌─────────────────┐

│ User Input │

└────────┬────────┘

│

▼

┌───────────────────────────────────────┐

│ Authentication │

└───────────────────┬───────────────────┘

│

▼

┌───────────────────────────────────────┐

│ Validation │

└───────────────────┬───────────────────┘

│

┌─────────┴─────────┐

│ │

▼ ▼

┌─────────────────────┐ ┌─────────────────────┐

│ Mood Processing │ │ User Processing │

└──────────┬──────────┘ └──────────┬──────────┘

│ │

▼ ▼

┌─────────────────────┐ ┌─────────────────────┐

│ Data Storage │ │ Profile Management │

└──────────┬──────────┘ └──────────┬──────────┘

│ │

└─────────────┬───────────┘

│

▼

┌───────────────────────────────────────┐

│ Data Analytics │

└───────────────────┬───────────────────┘

│

▼

┌───────────────────────────────────────┐

│ Data Visualization │

└───────────────────┬───────────────────┘

│

▼

┌───────────────────────────────────────┐

│ UI Rendering │

└───────────────────┬───────────────────┘

│

▼

┌───────────────────────────────────────┐

│ User Feedback │

└───────────────────────────────────────┘

```

### DFD Components Description

1. \*\*User Input\*\*: Entry point for all user actions including mood logging, settings changes, and data requests.

2. \*\*Authentication\*\*: Verifies user identity and manages login/logout processes.

3. \*\*Validation\*\*: Ensures all data meets the required format and constraints before processing.

4. \*\*Mood Processing\*\*: Handles specific mood-related operations such as logging, editing, and deleting mood entries.

5. \*\*User Processing\*\*: Manages user-specific operations such as profile updates and preference changes.

6. \*\*Data Storage\*\*: Interfaces with Local Storage to persist and retrieve data.

7. \*\*Profile Management\*\*: Handles user profile information, badges, and achievements.

8. \*\*Data Analytics\*\*: Processes raw mood data to generate statistics, correlations, and predictions.

9. \*\*Data Visualization\*\*: Transforms processed data into visual representations.

10. \*\*UI Rendering\*\*: Final presentation layer that displays interfaces to the user.

11. \*\*User Feedback\*\*: Communicates system status and operation results back to the user.

### Data Flow Descriptions

1. \*\*User → Authentication\*\*: User credentials flow to authentication process.

2. \*\*Authentication → Validation\*\*: Authenticated requests flow to validation.

3. \*\*Validation → Processing\*\*: Validated data flows to appropriate processing module.

4. \*\*Processing → Storage\*\*: Processed data flows to persistence layer.

5. \*\*Storage → Analytics\*\*: Stored data flows to analytical processing.

6. \*\*Analytics → Visualization\*\*: Processed statistics flow to visualization components.

7. \*\*Visualization → UI Rendering\*\*: Visual representations flow to rendering engine.

8. \*\*UI Rendering → User Feedback\*\*: Rendered content flows to user as feedback.

---

# 5. Screenshots

## 5.1 Frontend/Client/End User Side Screenshots

## 5.2 Backend/Admin Side Screenshots

## 5.3 Other Screenshots

---

# 6. Conclusion And Future Work

---

# REFERENCES