🌊 Ocean AI - *Your Journey to Calm*

Technical Design Document

1. Overview

Ocean AI is an AI-powered mental health journaling application designed to help users cultivate emotional awareness and mindfulness through daily reflection. The application combines traditional journaling practices with artificial intelligence to provide personalized prompts, emotional insights, and goal tracking.

The application follows a calming ocean-themed design philosophy, using soothing cyan and blue gradients to create a tranquil user experience that supports mental well-being.

2. Motivation

Many individuals struggle to maintain consistent journaling habits and often lack the tools to identify patterns in their emotional lives. Traditional journaling apps offer basic text entry but miss opportunities to provide meaningful insights or personalized guidance. Blank page paralysis remains a high friction point for many users who could benefit from daily journaling.

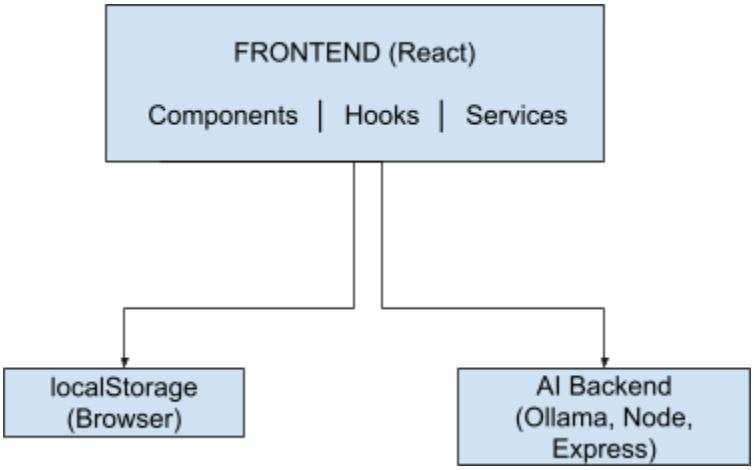
Ocean AI addresses these challenges by providing:

- AI-generated reflection prompts tailored to user context
- Automated sentiment analysis and pattern recognition
- Visual tracking of journaling streaks to encourage the habit
- Integrated goal setting and progress monitoring

3. Architecture Overview

3.1 High-Level Architecture

Ocean AI follows a modern client-server architecture with a React-based frontend and separate AI and backend services.



3.2 Component Hierarchy

Component	Responsibility
App	Root component; manages view state and orchestrates child components
NavBar	Navigation header with view switching (Write, Calendar, Goals, Insights)
JournalBook	Paginated display of journal entries with book-like UI metaphor
PromptCard	Displays AI-generated reflection prompts with regeneration capability
CalendarView	Monthly calendar showing journaling activity and streak tracking
GoalsView	Goal management interface with add, toggle, and delete functionality
InsightsView	AI-powered analytics dashboard showing sentiment, themes, and patterns

4. Technology Stack

4.1 Frontend Technologies

Technology	Version	Purpose
React	18.2.0	UI component library with hooks for state management
Vite	4.3.9	Build tool with fast HMR for testing
Tailwind CSS	3.3.2	Utility-first CSS framework for styling
Lucide React	0.263.1	Icon library for consistent UI iconography

4.2 AI + Backend Technologies

Technology	Version	Purpose
Node.js	19.0.0	JavaScript runtime for the backend server
Express.js	4.18.2	REST API framework handling the endpoints
Ollama	0.13.5	Local LLM inference for prompt generation and sentiment analysis

5. Data Flow and State Management

5.1 Custom Hook: useJournalData

The application centralizes all data operations through a custom React hook (`useJournalData`) that serves as the single source of truth for application state.

State Variables:

Variable	Type	Description
`entries`	Array	Journal entries with id, text, prompt, date, displayDate
`goals`	Array	Goals with id, text, createdDate, completed status
`insights`	Object	Cached AI analysis results
`isLoading`	Boolean	Initial data loading state
`isAnalyzing`	Boolean	AI operation progress flag

Exposed Methods:

Method	Description
`saveEntry(text, prompt)`	Creates and persists a new journal entry
`addGoal(text)`	Creates a new goal
`toggleGoal(id)`	Toggles goal completion status
`deleteGoal(id)`	Removes a goal
`generatePrompt()`	Fetches AI-generated reflection prompt
`analyzeEntries()`	Triggers AI analysis of all entries

5.2 Schemas

Journal Entry Schema

```
{
  id: number,           // Timestamp as unique identifier
  text: string,         // Journal entry content
  prompt: string,       // AI-generated prompt used (if any)
  date: string,         // ISO timestamp
  displayDate: string   // Human-readable formatted date
}
```

Goal Schema

```
{
  id: number,           // Timestamp as unique identifier
  text: string,         // Goal description
  createdDate: string,  // ISO timestamp
  completed: boolean    // Completion status
}
```

Insights Schema

```
{
  overallSentiment: string,  // "positive" | "negative" | "neutral"
  sentimentScore: number,    // Score from -1 to 1
  emotionalTrend: string,    // Trend description
  entryCount: number,        // Number of entries analyzed
  themes: string[],          // Recurring topics identified
  patterns: string[],        // Behavioral patterns noticed
  encouragement: string,     // Personalized supportive message
  suggestion: string         // Actionable recommendation
}
```

6. AI Integration

6.1 AI Service Architecture

The `aiService.js` module handles all communication with the AI backend through a clean interface.

```
class AIService {
  async generatePrompt(recentEntries, goals)  // Returns personalized prompt
  async analyzeEntries(entries)               // Returns insights object
}
```

6.2 API Endpoints

Endpoint	Method	Purpose
`/api/generate-prompt`	POST	Generates contextual reflection prompts based on recent entries and goals

Endpoint	Method	Purpose
`/api/analyze-entries`	POST	Performs sentiment analysis, theme extraction, and pattern recognition

6.3 Graceful Degradation

The AI service implements fallback behavior to ensure the application remains functional:

- Prompt generation falls back to default reflective questions like "What's one thing you'd like to process from today?"
- Analysis failures are surfaced to users with actionable error messages
- Cached insights remain accessible when network requests fail

7. Design Patterns

7.1 Service Layer Pattern

Business logic is encapsulated in dedicated service modules:

- `aiService.js`: Handles all AI backend communication
- `storageService.js`: Abstracts localStorage operations with async interface

7.2 Custom Hooks Pattern

The `useJournalData` hook centralizes state management and side effects, providing a clean API for components while handling data persistence automatically.

7.3 Constants Centralization

All magic values are centralized in `constants.js` so they can be used across multiple files without duplication.

8. Future Enhancements

Enhancement	Description
Authentication	User accounts with OAuth providers (Google, Apple)
Export Features	PDF/Markdown export of journal entries and insights reports
Attachments	Support images and other files to supplement entries (like clippings in a journal)
Notifications	Push notifications for journaling reminders and streak maintenance

9. Security Considerations

All data is stored locally in the browser. Journal entries are sent to the AI backend for prompt generation and analysis. When running Ollama locally, this means data never leaves the user's machine.

10. Performance Considerations

- Vite provides fast development builds for rapid testing
- Tailwind CSS removes unused styles (content attribute in config will allow it to scan those files and create a minimal CSS file)
- localStorage provides instant local data access