Mental Wellness App (Relax & Focus)

---

1. Problem Statement (Why This Architecture Exists)

Mental wellness applications often fail users for three main reasons:

1. **Too much pressure**
    - Mandatory accounts
    - Streaks and performance tracking
    - Long programs that feel overwhelming
2. **Lack of privacy**
    - Sensitive data stored on servers
    - Users unsure who can see their information
3. **Overengineering**
    - Complex backends for simple self-help needs
    - Heavy data collection without real user benefit

Our goal is to build a **mental wellness app that helps users calm their mind and regain focus**, while:

- remaining usable without login
- minimizing data collection
- keeping the system simple and ethical
- still being a complete full-stack application

This directly influences the chosen tech stack and architecture.

PART 1: TECHNOLOGY STACK

(Frontend, Backend, Database)

---

2. Frontend Stack (User-Facing Layer)

Technologies Used

- **Next.js (React)** – JavaScript

- HTML / CSS
- Tailwind CSS or CSS Modules
- Browser storage (localStorage / IndexedDB)

Problem the Frontend Solves

Users need:

- Immediate access to mental wellness tools
- A calm, low-friction interface
- Offline and private usage
- Short, flexible interactions

A heavy server-driven UI would slow users down and introduce privacy concerns.

How the Frontend Solves This

The frontend handles **most of the app's logic**, including:

- Relax Mode (breathing, grounding, visualization)
- Focus Mode (brain dump, next action, timers, planning)
- Temporary data storage (24–48 hours)
- Gentle UX flows without pressure

By keeping logic client-side:

- The app works even without internet
- Sensitive mental data stays on the user's device
- User experience remains fast and responsive

Key Frontend Responsibilities

- Render all UI screens
- Handle focus timers and Pomodoro sessions
- Manage temporary focus data
- Automatically delete old data
- Encrypt data before optional backup
- Provide optional login UI (not required)

---

3. Backend Stack (Support Layer)

Technologies Used

- **Node.js**
- **Express.js**
- JWT (authentication)
- bcrypt (password hashing)

Problem the Backend Solves

Some users want:

- Data continuity across devices
- Protection against accidental data loss

However, forcing all users to use accounts would:

- break trust
- increase friction
- contradict privacy-first design

How the Backend Solves This

The backend exists **only for optional features**, not core functionality.

It provides:

- Optional authentication
- Secure encrypted backup storage
- Data deletion on request

The backend **does not**:

- process mental wellness logic
- analyze user behavior
- store emotional or raw text data

This keeps the server's role minimal and ethical.

Backend Responsibilities

- Register and authenticate users (optional)
- Accept encrypted focus data from client
- Store encrypted backups
- Return backups on request
- Delete data when requested

If the backend goes down, the app **still works** locally.

---

4. Database Stack (Persistence Layer)

Technologies Used

- **MongoDB**
- Mongoose (ODM)

Problem the Database Solves

Only one problem:

> Securely storing optional backups for logged-in users

The database is **not** used for:

- activity tracking
- analytics
- user profiling
- long-term emotional history

Database Structure (Minimal)

Collections

- `users`
- `backups`

What is stored

- Encrypted focus-planning data
- Hashed authentication credentials

What is never stored

- Brain dump text
- Relaxation session data
- Visualization content
- Emotional notes

This reduces risk and complexity.

PART 2: ARCHITECTURE VIEW

(Client Side vs Server Side)

---

5. Client-Side Architecture

Client-Side Problem

Mental wellness tools require:

- trust
- privacy
- immediacy
- low cognitive load

Server-heavy designs can:

- introduce delays
- increase anxiety around data
- fail when offline

Client-Side Responsibilities

The client handles:

- All Relax Mode features
- All Focus Mode features

- Brain dump and task breakdown
- Pomodoro timers
- Temporary data storage
- Auto-cleanup after 24–48 hours
- Encryption before backup
- Optional login interactions

This ensures:

- privacy by default
- offline capability
- fast user experience

Why Temporary Local Storage Is Used

Permanent storage creates:

- guilt around unfinished tasks
- pressure to “keep up”
- attachment to productivity

Temporary storage:

- supports short-term clarity
- encourages fresh starts
- aligns with mental wellness goals

---

6. Server-Side Architecture

Server-Side Problem

Users who *choose* to log in need:

- secure backup
- cross-device access
- control over their data

At the same time, the server must:

- minimize sensitive data exposure
- avoid becoming a surveillance layer

Server-Side Responsibilities

The server:

- authenticates users (optional)
- stores encrypted backups
- never inspects mental data
- allows full data deletion

The server does **not**:

- enforce usage patterns
- analyze productivity
- influence user behavior

This maintains ethical boundaries.

---

7. Client–Server Interaction Flow

Without Login (Default)

- All data stays on device
- No server calls
- Full app functionality

With Login (Optional)

1. Client encrypts focus data
2. Encrypted data sent to server
3. Server stores encrypted blob
4. Client decrypts on retrieval

This design ensures:

- user control
- minimal trust requirements
- reduced breach impact

---

8. Deployment Overview

Client Side

- Hosted on **Vercel**
- Optimized for fast global access

Server Side

- Hosted on **Render / Railway / Heroku**

Database

- **MongoDB Atlas**

---

9. Why This Architecture Is Appropriate

This architecture:

- Solves real mental wellness problems
- Respects user privacy
- Avoids unnecessary complexity
- Is scalable without redesign
- Demonstrates full-stack competence

Most importantly:

> The app remains helpful even if the server disappears.

That is intentional.

---

10. Final Summary

This tech stack and architecture were chosen to support a **human-centered mental wellness app**, where:

- the frontend carries the experience
- the backend supports continuity, not control
- the database stores the minimum necessary data
- privacy and trust are first-class concerns

One-line Technical Philosophy

> Build the smallest system that helps the user — and no larger.

If you want, next I can:

- convert this into a **diagram**
- shorten it into a **2-page technical brief**
- rewrite it for **interview or viva**
- help you prepare **justifications for each decision**

Just tell me what you want next.