PHASE 1 — Homepage Module (SSR)

Phase Goal

Build a **calm, fast, server-side rendered homepage** that:

- explains the app clearly
- builds trust
- routes users to Relax, Focus, Resources, or Login
- works without user data or backend dependency

---

1. Phase 1 – Technical Decisions (Lock First)

Before coding, lock these:

- [ ] Homepage will be **Server-Side Rendered (SSR)**
- [ ] No user data used
- [ ] No API calls required
- [ ] No authentication logic
- [ ] No local storage usage
- [ ] Static + calm content only

✅ Once locked, **do not change during Phase 1**.

---

2. Project Setup (Homepage Scope Only)

2.1 Initialize Frontend

- [ ] Create Next.js project (JavaScript)
- [ ] Clean default boilerplate

[ ] Set up basic folder structure:

```
/app
  page.js      // Homepage (SSR)
  layout.js
/components
/styles
```

-

2.2 Global Layout & Theme

- [ ] Set global font (soft, readable)
- [ ] Set background gradient / colors
- [ ] Ensure mobile responsiveness
- [ ] Remove unnecessary animations

Goal: **visual calm**

---

3. Homepage UI Components (Build One by One)

3.1 Header / Navigation Component

- [ ] App logo / name
- [ ] Navigation links:
    - Focus
    - Relax
    - Resources
    - Login / Signup
- [ ] Make Login visually secondary

Constraint:

> No dropdowns, no popups, no modals.

3.2 Hero Section

- [ ] Main headline
  “Your Sanctuary for Focus and Calm”
- [ ] One-line description (soft, human tone)
- [ ] Primary CTA button:
    - “Begin Your Journey”

SSR Requirement:

> Text must render fully from server.

3.3 Secondary Low-Commitment CTA

- [ ] Add secondary button:
    - “Try a 2-minute calm session”
    - or “Start without signing up”
- [ ] Lower visual weight than primary CTA

Purpose:

> Reduce fear of commitment.

3.4 Relax vs Focus Explainer Section

- [ ] Two simple blocks/cards:
    - Relax Mode — calm your mind in minutes
    - Focus Mode — start tasks without pressure
- [ ] Use icons (optional)
- [ ] Max one line each

No paragraphs. No feature lists.

3.5 Trust & Privacy Message

- [ ] Add subtle trust copy:
    - “Private by design · No tracking · Free to use”
- [ ] Small font, low contrast emphasis

Purpose:

> Build trust before the first click.

---

4. Routing & Navigation Logic

- [ ] “Begin Your Journey” routes to Focus or choice screen
- [ ] Secondary CTA routes directly to Relax Mode
- [ ] Nav links route correctly (pages can be placeholders)

At this phase:

- Pages can exist as **empty placeholders**
- Only routing matters

---

5. Server-Side Rendering Verification

- [ ] Confirm homepage renders on server
- [ ] Disable JS temporarily → content still visible
- [ ] View page source → HTML contains content
- [ ] No hydration errors

This confirms **true SSR**, not accidental CSR.

---

6. Non-Functional Checks (Phase 1)

Performance

- [ ] Page loads instantly
- [ ] No layout shifts
- [ ] Minimal JS bundle

Accessibility

- [ ] Readable contrast
- [ ] Clear headings
- [ ] Keyboard navigation works

UX Safety

- [ ] No aggressive copy
- [ ] No forced signups
- [ ] No popups

---

7. Phase 1 Exit Criteria (Very Important)

You **do not move to Phase 2** until:

- [ ] Homepage fully SSR
- [ ] Calm, complete visual flow
- [ ] All CTAs route correctly
- [ ] No backend dependency
- [ ] No unfinished UI sections

If any of these fail → **fix before moving on**.

---

8. Phase 1 Deliverable (What You Should Have)

At the end of Phase 1, you should have:

✅ A production-ready homepage

✅ Server-side rendered

✅ Calm, trustworthy UX

✅ Clear entry paths

✅ Solid foundation for next modules

PHASE 2 — Relax Mode Module (Client-Side)

Phase Goal

Build a fully functional **Relax Mode** that provides immediate calming experiences without requiring login, backend access, or data storage.

This phase focuses on:

- real-time interaction
- emotional safety
- simplicity
- zero persistence

---

1. Phase 2 – Technical Decisions (Lock First)

Before implementation, lock the following:

- Relax Mode is **client-side only**
- No backend calls
- No local storage
- No analytics or tracking
- Sessions are ephemeral
- User can exit at any time without consequence

These constraints must not be violated during Phase 2.

---

2. Routing & Page Setup

Create Relax Mode route:
/relax

-
- Ensure navigation from Homepage works
- Page should load instantly without backend dependency

---

3. Relax Mode Landing Screen

Tasks

- Design a minimal entry screen
- Display available relaxation options:
    - Breathing
    - Visualization
    - Grounding (finger tapping)
- Keep text minimal and calming
- Avoid instructions longer than one line

Exit Criteria

- User understands what to do within 3 seconds
- No cognitive overload

---

4. Breathing Session Implementation

Functionality

- Implement a simple breathing cycle
- Use visual cues (expand and contract)
- Optional audio guidance
- Session length: 2–5 minutes

UX Rules

- Start immediately
- Allow exit at any time
- No timers shown upfront
- No completion messages that imply success or failure

---

5. Visualization Session Implementation

Functionality

- Audio-guided visualization
- Neutral and inclusive imagery
- Slow pacing

Constraints

- No text input
- No personal reflection prompts
- No saved state

---

6. Grounding / Finger Tapping Session

Functionality

- Guided rhythmic tapping
- Visual or audio prompts
- Simple repetition

Purpose

- Support users who struggle with breathing exercises

- Provide body-based grounding

---

7. Session Exit & Closure

Tasks

- Add gentle end-of-session messaging
- Examples:
    - “Take your time.”
    - “You can return whenever you need.”
- No progress indicators
- No rewards or streaks

---

8. Failure & Edge Case Handling

- If audio fails, fallback to visual guidance
- If user navigates away, session ends silently
- If page refreshes, no recovery needed

---

9. Non-Functional Requirements

Performance

- Immediate interaction
- No noticeable lag

Accessibility

- Avoid flashing visuals
- Calm color contrast
- Readable text size

---

10. Phase 2 Exit Criteria

You should not move forward until:

- Relax Mode works fully offline
- All three techniques function independently
- No data is stored anywhere
- UX remains calm and interruption-free

Phase 2 Deliverable

At the end of this phase, you will have:

- A complete Relax Mode
- Zero backend dependency
- Immediate user value
- Strong emotional trust with users

PHASE 3 — Focus Mode Module

Phase Goal

Build a **gentle, psychology-informed Focus Mode** that helps users start tasks, reduce mental overload, and work in short, non-pressured sessions.

This phase must:

- reduce task-initiation friction
- avoid guilt, scoring, or productivity pressure
- store data temporarily and safely
- work fully without login

---

1. Phase 3 – Technical Decisions (Lock First)

Before implementation, lock these decisions:

- Focus Mode is **client-side first**
- No mandatory backend dependency
- All data stored **locally with TTL**
- Brain Dump data is never backed up

- No streaks, no metrics, no gamification
- User can stop at any time without penalty

These rules define the entire phase.

---

2. Routing & Page Setup

Create Focus Mode route:
/focus

-
- Ensure homepage navigation routes correctly
- Page must load even if backend is unavailable

---

3. Focus Mode Landing Screen

Tasks

- Create a simple entry screen with 3 options:
    - Clear my head (Brain Dump)
    - Start a focus session
    - Plan for today / tomorrow
- Avoid suggesting a “correct” order
- Keep copy minimal and reassuring

Exit Criteria

- User understands options immediately
- No forced sequence

---

4. Brain Dump Feature

Purpose

Reduce mental overload by externalizing thoughts.

Tasks

- Add free-text input area
- Allow unlimited text
- No validation
- Provide a clear “done” or “close” option

Data Rules

- Store only in memory or short-lived local storage
- Auto-clear when session ends or after short TTL
- Never include in backups

Exit Criteria

- Text disappears safely after session
- No history or recovery

---

5. Next Physical Action Feature

Purpose

Solve “I don’t know where to start” paralysis.

Tasks

- Input field for a single task
- Prompt user to define the next visible action
- Show example hints
- Prevent vague inputs where possible

Data Handling

- Store locally with timestamp
- Eligible for encrypted backup later

---

6. Time-Boxing / Pomodoro Feature

Purpose

Shift focus from outcomes to presence.

Tasks

- Timer options: 10, 15, 25 minutes
- Start / stop controls
- Gentle completion message
- No automatic continuation

UX Rules

- Stopping early is allowed
- No failure state
- No score or streak

---

7. Gentle Planning Feature

Purpose

Reduce future decision fatigue.

Tasks

- Allow user to plan 1–3 tasks
- Break tasks into small time blocks
- Allow marking blocks as done or not done

Data Rules

- Store locally
- Attach TTL (24–48 hours)
- Auto-expire silently

---

8. Self-Compassion Closure

Purpose

Separate self-worth from productivity.

Tasks

- Display gentle closure messages
- Avoid performance language
- Encourage rest without guilt

---

9. Local Storage & Cleanup Logic (Focus-Specific)

Tasks

- Implement centralized storage utility
- Timestamp all entries
- Run cleanup on:
    - App load
    - Focus Mode entry

Cleanup Rules

- Expired data is deleted silently
- No alerts or warnings

---

10. Failure & Edge Case Handling

Scenario	Expected Behavior
User exits mid-session	Session ends silently
Browser refresh	Data persists only if valid
Storage cleared	App continues without error
Backend unavailable	No impact

---

11. Non-Functional Requirements

Performance

- Immediate interaction
- No heavy computations

Accessibility

- Simple language
- Large touch targets
- Calm visuals

Privacy

- No tracking
- No analytics
- No external calls

---

12. Phase 3 Exit Criteria

Do not proceed until:

- All focus tools work independently
- Data expires correctly
- Brain dump never persists
- UX feels non-judgmental
- Focus Mode works offline

Phase 3 Deliverable

At the end of this phase, you will have:

- A complete Focus Mode
- Temporary, privacy-safe data handling
- Psychology-aligned productivity tools
- A strong core product

PHASE 4 — Local Storage, Cleanup, and Data Safety Layer

Phase Goal

Build a **robust, centralized local data layer** that:

- stores focus-related data temporarily
- enforces automatic expiration (TTL)
- protects privacy by default
- works consistently across the app

This phase ensures the app is **safe, predictable, and resilient** before introducing backend features.

---

1. Phase 4 – Technical Decisions (Lock First)

Lock these rules before implementation:

- All user-generated focus data is temporary by default
- Data expiration is automatic and silent
- No alerts or warnings on deletion
- Brain Dump data is excluded from persistence
- No backend dependency
- No analytics or telemetry

These constraints must remain unchanged.

---

2. Define Data Categories

Clearly separate data types to avoid accidental persistence.

Data Types

1. **Session-only data**
    - Brain dump text
    - Active timers
    - In-progress focus sessions
2. **Temporary local data**
    - Next physical actions
    - Planned tasks
    - Time blocks and completion status

---

3. Storage Utility Design

Tasks

- Create a single storage utility module
- All reads and writes must go through this module
- No direct `localStorage` usage elsewhere

Required Functions

- `save(key, data)`
- `load(key)`
- `remove(key)`
- `cleanupExpired()`

---

4. Timestamp & TTL Strategy

Tasks

- Attach `createdAt` timestamp to every stored object
- Define TTL rules:
    - Default: 24 hours
    - Optional extension: up to 48 hours
- Compare timestamps on load

Rule

If data is expired:

- Delete immediately
- Return `null`
- Do not notify the user

---

5. Cleanup Execution Points

Tasks

Run cleanup logic at the following points:

- On app load
- On entering Focus Mode
- After completing a focus session

Cleanup must be:

- silent
- fast
- non-blocking

---

6. Error & Corruption Handling

Tasks

- Wrap storage access in try/catch
- Handle corrupted JSON gracefully
- Fallback to empty state on failure

Rule

Never crash the app due to storage errors.

---

7. Privacy & Safety Guarantees

Enforced Guarantees

- No long-term history
- No hidden retention
- No background syncing
- No recovery of expired data

This must be explicitly true in code.

---

8. Testing Scenarios (Manual)

Test the following before proceeding:

- Data disappears after TTL
- App works after storage is cleared
- Refreshing browser behaves correctly
- Brain dump is never persisted
- Cleanup does not affect Relax Mode

---

9. Non-Functional Requirements

Performance

- Cleanup runs in under a few milliseconds
- No UI blocking

Reliability

- Works across browser sessions
- No dependency on network

---

10. Phase 4 Exit Criteria

You must not proceed unless:

- All temporary data expires correctly
- No data survives beyond TTL
- No crashes from corrupted storage
- Storage logic is centralized and reusable

Phase 4 Deliverable

At the end of this phase, you will have:

- A safe, predictable data layer
- Strong privacy guarantees
- Clean separation of temporary vs session data
- A stable foundation for backups and auth

PHASE 5 — Resources Module (Backend-Driven Content)

Phase Goal

Build a **backend-driven Resources Module** that serves curated, trustworthy external support content to the frontend while maintaining:

- zero user tracking
- zero personalization
- zero sensitive data handling

This phase validates backend integration **without touching auth or user data**.

---

1. Phase 5 – Technical Decisions (Lock First)

Before implementation, lock these decisions:

- Resources are **read-only**
- Backend only serves content
- No user identifiers in requests
- No analytics on resource usage
- Frontend treats content as informational only

These rules must not change in this phase.

---

2. Backend Setup (Minimal Scope)

Tasks

- Initialize Node.js + Express server

Set up project structure:

```
/server
  /routes
  /controllers
  /data
  server.js
```

-
- Enable CORS for frontend domain
- Add basic health-check endpoint

Backend should be simple and stable.

---

3. Resource Data Definition

Tasks

- Define resource categories:
    - Helplines
    - Educational content
    - Self-help references
- Decide data source:
    - Static JSON file (recommended for Phase 5)
    - MongoDB (optional, can be added later)

Data Rules

- No dynamic personalization
- No user-specific filtering
- Content must be human-reviewed

---

4. Resources API Design

Endpoint

GET /api/resources

Tasks

- Implement controller to return resource data
- Structure response clearly by category
- Include disclaimer text in response

Example Response Structure

{"helplines":[],"education":[],"selfHelp":[],"disclaimer":"This app is not a substitute for professional mental health care."}

---

5. Frontend Integration (Resources Page)

Tasks

- Create `/resources` page
- Fetch data from backend on page load
- Render categories clearly
- Mark all links as external
- Handle loading and error states calmly

UX Rules

- No urgency language
- No auto-redirects
- No popups

---

6. Failure & Fallback Handling

Tasks

- If backend is unreachable:
  - Show static fallback resources
  - Display a gentle message
- If a link fails:
  - Display non-alarming error
- Never block access to the page

---

7. Security & Privacy Guarantees

Enforce the following:

- No request parameters tied to user behavior
- No cookies required

- No logging of IP or access patterns (beyond basic server logs)
- HTTPS-only communication

---

8. Non-Functional Requirements

Performance

- Small payload size
- Fast response time

Maintainability

- Easy to update resource list
- No frontend redeploy required for content updates

---

9. Testing Checklist

Before proceeding:

- Backend returns correct data
- Frontend renders all categories
- Page works without login
- App behaves correctly if backend is down
- No console errors

---

10. Phase 5 Exit Criteria

You must not proceed unless:

- Backend is stable and isolated
- Resources page loads correctly
- No user data is involved
- Fallback behavior works as intended

Phase 5 Deliverable

At the end of this phase, you will have:

- A working backend service
- A dynamic Resources page
- First real frontend–backend integration
- Confidence to proceed to authentication

PHASE 6 — Login / Sign-Up & Authentication Module

Phase Goal

Build an **optional authentication system** that enables:

- encrypted backups
- cross-device continuity

while ensuring:

- zero impact on core app usage
- minimal user data collection
- clear consent and easy exit

Authentication must **support the product**, not control it.

---

1. Phase 6 – Technical Decisions (Lock First)

Before implementation, lock these decisions:

- Login is optional
- Core features work without authentication
- No social login in Phase 1
- Email + password only
- Authentication exists only for backup purposes
- No personalization or tracking tied to auth

These constraints define the entire phase.

---

2. Backend: Authentication Setup

Tasks

Add authentication routes:
POST /auth/register
POST /auth/login

-
- Install required libraries:
    - bcrypt (password hashing)
    - jsonwebtoken (JWT)
- Configure environment variables for secrets

---

3. User Model Design

Tasks

- Create `User` model in MongoDB
- Fields:
    - email
    - passwordHash
    - createdAt

Constraints

- No profile fields
- No mental health metadata
- No usage statistics

---

4. Registration Flow

Backend Tasks

- Validate email and password
- Hash password using bcrypt
- Store user record
- Return success response

Frontend Tasks

- Create sign-up UI

- Explain why login exists
- Explicitly state that login is optional
- Provide “Continue without login” option

---

5. Login Flow

Backend Tasks

- Verify credentials
- Generate JWT token
- Return token to client

Frontend Tasks

- Store token securely (in memory or secure storage)
- Handle login errors gracefully
- Avoid aggressive retry prompts

---

6. Token Management

Tasks

- Attach JWT token to authenticated requests
- Use short-lived tokens
- Handle token expiration silently

Rule

If token expires:

- User is logged out
- App continues to work locally

---

7. Account Deletion Flow

Tasks

- Add account deletion endpoint
- Remove user record
- Remove all associated backups
- Invalidate tokens

UX Rule

Deletion must be:

- Easy to find
- Clearly explained
- Immediate

---

8. Failure & Edge Case Handling

Scenario	Expected Behavior
Login fails	Clear error, retry allowed
Backend unavailable	App usable without login
Token expired	Silent logout
User deletes account	Data removed immediately

---

9. Security & Privacy Guarantees

Enforce the following:

- Passwords never stored in plaintext
- JWT secrets stored securely
- HTTPS only
- No auth-based analytics
- No third-party identity providers

---

10. Non-Functional Requirements

Security

- Strong hashing
- Secure token handling

Usability

- Calm language
- No forced prompts
- No blocking flows

---

11. Phase 6 Exit Criteria

Do not proceed unless:

- Login is fully optional
- Auth failures do not break app
- Users can delete account and data
- No sensitive data leaks

Phase 6 Deliverable

At the end of this phase, you will have:

- A working authentication system
- Clear consent-based login
- Minimal, privacy-safe user records
- A stable base for backups

PHASE 7 — Encrypted Backup & Restore System

Phase Goal

Add an **optional, privacy-preserving backup system** that allows users who choose to log in to:

- back up focus-related data securely

- restore data across devices
- retain full control over deletion

This phase must **not weaken** the privacy guarantees established earlier.

---

1. Phase 7 – Technical Decisions (Lock First)

Before implementation, lock these decisions:

- Backups are **optional**
- All encryption happens **on the client**
- Backend stores **only encrypted blobs**
- Brain Dump data is **never backed up**
- One active backup per user
- User can delete backups at any time

These rules must not change.

---

2. Define What Can Be Backed Up

Eligible Data

- Next physical actions
- Gentle plans (tasks and time blocks)
- Focus session metadata (non-sensitive)

Explicitly Excluded

- Brain dump text
- Relax mode usage
- Any emotional or reflective content

---

3. Client-Side Encryption Layer

Tasks

- Implement encryption utility using Web Crypto API

- Generate or derive encryption key on client
- Encrypt backup payload before sending
- Decrypt payload after restore

Rules

- Encryption key never sent to backend
- Backend cannot read user data
- Encryption failures do not block app usage

---

4. Backup Payload Design

Tasks

- Define a single backup payload structure
- Include:
    - encrypted data blob
    - version number
    - timestamp

Rule

Payload must be self-contained and replaceable.

---

5. Backend: Backup API

Endpoints

POST /backup
GET /backup
DELETE /backup

Tasks

- Authenticate requests using JWT
- Store encrypted payload in MongoDB
- Overwrite existing backup on new upload
- Return encrypted payload on request

Constraints

- No inspection of payload
- No partial updates
- No analytics

---

6. Frontend: Backup Flow

Backup Creation

- Trigger backup manually or automatically (opt-in)
- Encrypt data
- Send to backend
- Confirm success gently (no pressure)

Restore Flow

- Fetch encrypted payload
- Decrypt on client
- Restore to local storage
- Respect TTL rules

---

7. Deletion & Control

Tasks

- Allow user to delete backup explicitly
- Allow account deletion to cascade delete backups
- Confirm deletion clearly

Rule

Deletion must be immediate and irreversible.

---

8. Failure & Edge Case Handling

Scenario	Expected Behavior
Backup fails	Local data preserved
Restore fails	User continues locally
Key mismatch	Prompt reset option
Backend unavailable	Backup temporarily unavailable

---

9. Security & Privacy Guarantees

Enforce the following:

- Encrypted-at-rest backups
- Backend has zero knowledge of content
- No background syncing
- Explicit user consent for backup

---

10. Non-Functional Requirements

Performance

- Encryption fast enough to feel instant
- Small payload size

Reliability

- Safe overwrite behavior
- No silent corruption

---

11. Phase 7 Exit Criteria

Do not proceed unless:

- Encrypted backups work end-to-end
- Restore works across devices

- Brain dump is never backed up
- Backup deletion is reliable
- Core app works without backup enabled

Phase 7 Deliverable

At the end of this phase, you will have:

- A complete, privacy-first backup system
- Optional cross-device continuity
- Strong user trust guarantees
- A fully realized full-stack application

---

1. Development Methodology and Execution Policy

1.1 Phased Development Requirement

The system **shall be developed in clearly defined phases**, as outlined in the approved development roadmap. Each phase represents a logically complete and self-contained unit of functionality.

The development process **shall strictly follow a sequential, phase-based approach**. Parallel implementation of features across multiple phases is **not permitted**.

1.2 Feature-Level Development Constraint

Within each phase, the system **shall be implemented feature by feature**.

The developer **shall not begin implementation of a new feature** until the current feature has been fully completed and validated.

A feature shall be considered *complete* only when:

- The feature's functional requirements are fully implemented.
- The feature conforms to the system design constraints defined for that phase.
- The feature operates correctly without relying on functionality from future phases.
- The feature has been manually verified for correctness and stability.

1.3 Phase Completion Criteria

A phase shall be considered *complete* only when **all features within that phase** have been fully implemented and verified.

The developer **shall not proceed to the next phase** unless:

- All features in the current phase are complete.
- The defined phase exit criteria are satisfied.
- No unresolved defects remain that affect system stability or design guarantees.

1.4 Prohibited Development Practices

The following practices are explicitly prohibited:

- Partial implementation of features across phases.
- Skipping phases in the development roadmap.
- Implementing future-phase dependencies prematurely.
- Introducing backend, authentication, or persistence logic outside the designated phase.

Such practices may compromise system correctness, privacy guarantees, and maintainability.

1.5 Rationale

This structured, sequential development methodology ensures:

- Clear separation of concerns.
- Reduced cognitive and technical complexity.
- Preservation of privacy-first and client-first design principles.
- Improved testability and maintainability.
- Alignment with the approved system design and architectural decisions.

1.6 Compliance

All developers working on this system **must adhere to this development methodology**.

Deviation from this process requires explicit design review and approval.