Developer: Manjot Kaur

Bug and Refactoring report

Overview

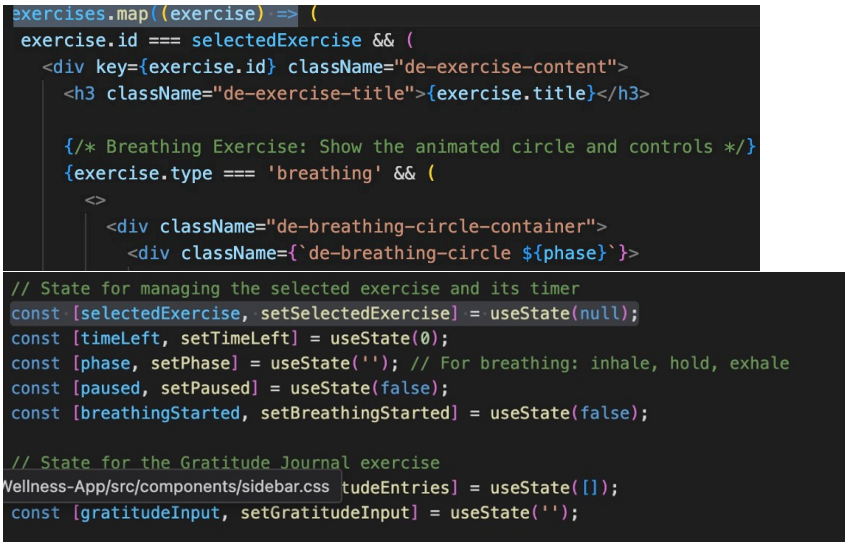
This document outlines the design issues and code smells identified in the ITR 3 user story: DailyExercises, the refactorings applied to address them, and the resulting improvements. Additionally, it addresses the resolution of reported bugs (BUG-015, BUG-004, BUG-003) raised by QA team. The focus is on three design-level refactorings that significantly improve the code structure and maintainability, ensuring they are not trivial changes like renaming methods but instead address critical design flaws.

Code Smells and Design Issues Resolved:

1. God Component and Monolithic Hook (Large Class and Long Method Smells)

Before:

- DailyExercises.jsx was a God Component, managing all six exercises in one file. The use DailyExercisesLogic hook in dailyExercisesService.js was a 200+ line God Hook, handling state and logic for all exercises (e.g., timeLeft, gratitudeEntries), violating SRP and making maintenance hard.
- *Impact*: Hard to debug, extend, or test.
- *Snippets*: Shows inline conditionals for rendering exercises and shows the bloated useDailyExercisesLogic hook with state for all exercises.



Developer: Manjot Kaur

Refactoring: Extract Exercise-Specific Components and Split God Hook

- Split DailyExercises.jsx into components like BoxBreathing, GratitudeJournal, etc.
- Divided useDailyExercisesLogic into hooks like useBreathingLogic, useGratitudeLogic, etc.
- Added ExerciseComponents mapping for dynamic rendering.

After

- Components and hooks are now focused (e.g., useBreathingLogic manages only Box Breathing state).
- *Benefits*: Improved modularity, easier debugging (fixed music bug), and better testability.
- *Snippets***:** Shows the dynamic rendering using ExerciseComponents mapping, highlighting the modularity and the simplified DailyExercises component using ExerciseComponent for rendering.

```jsx
// mapping exercies to their respective components
const ExerciseComponents = {
  breathing: BoxBreathing,
  gratitude: GratitudeJournal,
  grounding: GroundingExercise,
  'self-compassion': SelfCompassionExercise,
  trigger: TriggerIdentification,
  mindfulness: MindfulnessTimer,
};
```

```jsx
const DailyExercises = () => {
  const {
    selectedExercise,
    startExercise,
    goBack,
    audioRef,
    upliftingAudio,
  } = useDailyExercisesLogic();

  const selectedExerciseData = exercises.find((ex) => ex.id === selectedExercise);
  const ExerciseComponent = selectedExerciseData
    ? ExerciseComponents[selectedExerciseData.type]
    : null;

  // Renders either the exercise list or the selected exercise's focused view
  return (
    <div className="de-container">
      <Sidebar />
      <div className="de-main">
        <div className="de-content">
          <section className="de-section">
            <h2 className="de-section-title">Your Daily Wellness Boost</h2>
            {selectedExercise ? (
              // Focused view for the selected exercise
              <div className="de-focused-view">
                <button className="de-back-button" onClick={goBack}>
                  Back
                </button>
                {ExerciseComponent && (
                  <ExerciseComponent
                    exercise={selectedExerciseData}
                    goBack={goBack}
                    audioRef={audioRef}
                    upliftingAudio={upliftingAudio}
                  />
                )}
              </div>
            ) : (
              // Displays a card for each exercise,
              <div className="de-cards">
                {exercises.map((exercise) => {
                  const IconComponent = exercise.icon;
                  return (
```

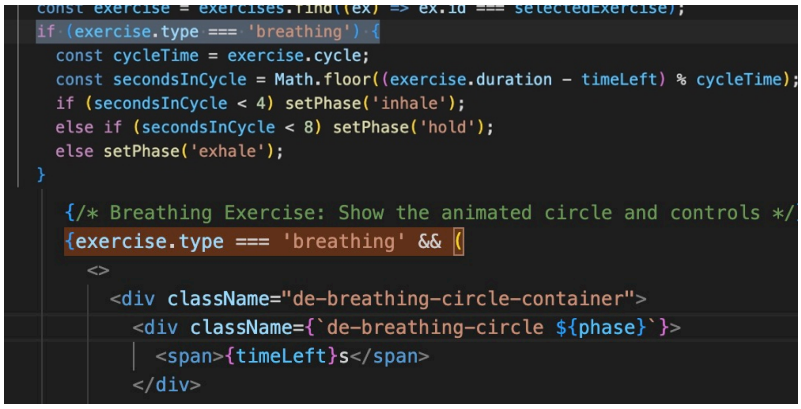
```jsx
const BoxBreathing = ({ exercise, goBack }) => {
  const { timeLeft, phase, paused, breathingStarted, startBreathingExercise, pauseExercise } =

  //Circular clock for breathing
  return (
    <div className="de-exercise-content">
      <h3 className="de-exercise-title">{exercise.title}</h3>
      <div className="de-breathing-circle-container">
        <div className={`de-breathing-circle ${phase}`}>
          <span>{timeLeft}s</span>
        </div>
        {breathingStarted && paused && <p className="de-phase">Paused</p>}
        {breathingStarted && !paused && <p className="de-phase">{phase}</p>}
      </div>
      <div className="de-controls">
        {!breathingStarted ? (
          <button className="de-start-button" onClick={startBreathingExercise}>
            Start
          </button>
        ) : (
          <>
            <button className="de-pause-button" onClick={pauseExercise}>
              {paused ? 'Resume' : 'Pause'}
            </button>
            <button className="de-done-button" onClick={goBack}>
              Done
            </button>
          </>
        )}
```

Developer: Manjot Kaur

2. String-Based Logic for Exercise Types (Primitive Obsession Smell)

Before:

- Used string comparisons (e.g., if (selectedExercise === 'breathing')) for exercise types and state management, leading to scattered string literals.
- *Impact*: Error-prone (typos), hard to maintain, and not scalable.
- *Snippets*: Shows the string-based conditional for rendering the breathing exercise UI and shows string-based logic in useDailyExercisesLogic for setting states.



Refactoring: Replace String Comparisons with Configuration Objects

- Added exerciseConfig object to centralize exercise type configurations (e.g., initialState, resetState).
- Replaced string conditionals with config lookups (e.g., exerciseConfig.breathing. initialState).
- Used ExerciseComponents mapping for rendering.

After:

- exerciseConfig centralizes configurations, no more string comparisons.
- *Benefits*: Scalable (adding exercises is easier), readable, and less error prone.
- *Snippets*: Shows the exerciseConfig object used to replace string comparisons. Shows useBreathingLogic using exerciseConfig for state initialization.

Developer: Manjot Kaur

```
// Configuration object for exercise types
export const exerciseConfig = {
  breathing: {
    initialState: {
      timeLeft: (exercise) => exercise.duration || 0,
      phase: 'inhale',
      paused: false,
      started: false,
    },
    resetState: {
      timeLeft: (exercise) => exercise.duration || 0,
      phase: 'inhale',
      paused: false,
      started: false,
    },
  },
```

```
// Hook for Box Breathing logic
export const useBreathingLogic = (exercise) => {
  const [timeLeft, setTimeLeft] = useState(exerciseConfig.breathing.initialState.timeLeft(exercise));
  const [phase, setPhase] = useState(exerciseConfig.breathing.initialState.phase);
  const [paused, setPaused] = useState(exerciseConfig.breathing.initialState.paused);
  const [breathingStarted, setBreathingStarted] = useState(exerciseConfig.breathing.initialState.started);

  const startBreathingExercise = () => {
    setBreathingStarted(true);
  };
```

3. Tightly Coupled Audio Logic in Mindfulness Timer (Feature Envy Smell)

Before:

- Audio logic for MindfulnessTimer was in useDailyExercisesLogic, despite being irrelevant to other exercises.
- *Impact*: Violated SRP, caused music bug (no playback), and hurt user experience.
- *Snippets*: Shows the audio logic embedded in useDailyExercisesLogic, illustrating the tight coupling.

```
useEffect(() => {
  if (audioRef.current && mindfulnessStarted) {
    if (playMusic && !paused) {
      audioRef.current.play();
    } else {
      audioRef.current.pause();
    }
  }
}, [playMusic, mindfulnessStarted, paused]);
```

Refactoring: Move Audio Logic to Mindfulness-Specific Hook

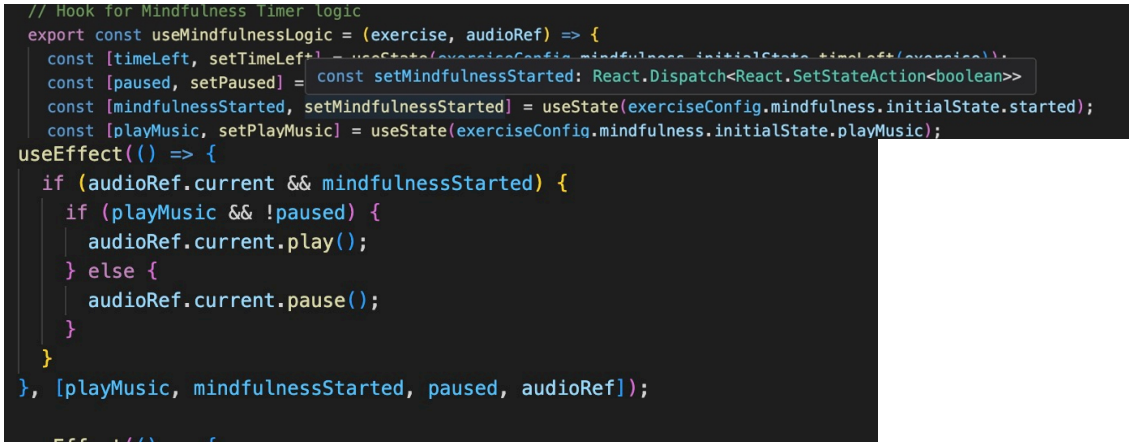
- Moved audio logic to useMindfulnessLogic, making it responsible for audioRef, playMusic, and playback.
- Updated useMindfulnessLogic to handle playback directly with audioRef.

After:

- useMindfulnessLogic now owns audio logic; useDailyExercisesLogic only provides audioRef.
- *Benefits*: Better encapsulation, fixed music bug, and improved user experience.

Developer: Manjot Kaur

- *Snippets*: Shows the audio logic now encapsulated in useMindfulnessLogic.



Bug Resolution Context

Resolved Bugs:

- **BUG-015 (User Not Informed of Errors During Data Fetching)**:
  - **Component**: FindHelp
  - **Reported By**: Ammar Faisal
  - **Date Reported**: March 24, 2025
  - **Issue**: Users were not informed of errors (e.g., API not loaded), leading to a frozen UI and poor user experience.
  - **Date Resolved:** March 29, 2025
  - **Resolved by**: Manjot Kaur
  - **Resolution Description**: The FindHelp component handle the API fetch bug through error management: it validates the API key, displaying a message if missing, and uses setError to show user-friendly messages for invalid postal codes, API failures, or no results. The loadError state catches Google Maps API script loading issues, while mock data ensures a fallback if API calls fail, maintaining a seamless user experience.

- **BUG-004 (Survey Responses Not Saved if User Navigates Away)**:
  - **Component**: Survey Check-In
  - **Reported By**: Ammar Faisal
  - **Date Reported**: March 24, 2025
  - **Issue**: Survey responses were lost if the user navigated away mid-check-in.
  - **Date Resolved:** March 29, 2025
  - **Resolved by**: Manjot Kaur

    - **Resolution Description**: The reported issue is actually not a bug, and it's an intended design. Also, since the survey is so short (7 questions , 10-15 seconds), there is really no reason to tie up browser resources storing the survey data. The solution for the proposed issue, might make sense for a longer survey, but doesn't really do much for such a short survey.

- **BUG-003 (Wellness Index Not Displayed After Survey Completion)**:
    - **Component**: Survey Completion / Wellness Index
    - **Reported By**: Ammar Faisal
    - **Date Reported**: March 24, 2025
    - **Issue**: The wellness index score didn’t display after survey submission due to a failure in fetchMostRecentWellnessIndex
    - **Date Resolved:** March 29, 2025
    - **Resolved by**: Manjot Kaur
    - **Resolution**: The issue was resolved by implementing error handling in fetchMostRecentWellnessIndex, which catches Appwrite backend errors and returns null if the fetch fails. The Survey.jsx component then uses a conditional check on wellnessIndex to display a message based on the score or defaults to "Loading your wellness index…" if null, ensuring the UI doesn’t stay stuck indefinitely.