I have analyzed your `package.json`. It is a very modern stack (React Native 0.79, React 19, React Navigation 7).

Since you are using **React Native 0.79 (Fabric/New Architecture)** and specifically requested a solution that preserves perfect styling without manual CSS hacking, the best approach is to use a **Hybrid "Streamer" Architecture**.

You already have the critical libraries installed:

1. `@dr.pogodin/react-native-fs` (File System)
2. `react-native-zip-archive` (Unzipping)
3. `react-native-webview` (Rendering engine)
4. `@react-native-async-storage/async-storage` (Progress saving)

Below is the **Developer Implementation Document**. You can copy this content into a PDF or a `.md` file for your team.

Technical Specification: Reactive EPUB Reader Implementation

Project: LifeInTheUK **RN Version:** 0.79.3 (Fabric) **Date:** November 2025

1. Executive Summary

This document outlines the implementation of a high-fidelity EPUB reader. The goal is to render books with 100% style accuracy (images, fonts, layouts) while managing "Business Logic" (Paywalls, Navigation, Progress) in the native layer.

Core Concept: We will not parse HTML natively. Instead, we will:

1. **Unzip** the EPUB file to the device's local storage.
2. **Serve** the content to a `WebView` via the file system.
3. **Inject** the `epub.js` library into the WebView to handle pagination and rendering.
4. **Bridge** communication to React Native to handle Chapter Locking (Paywall) and Progress saving.

---

2. Architecture & Data Flow

Kod snippet'i

```
graph TD
    A[User Selects Book] -->|Native| B(Check File System)
    B -->|If zip| C[Unzip via ZipArchive]
    B -->|If unzipped| D[Mount WebView]
```

```
    D -->|Load HTML Skeleton| E[WebView Engine]
    E -->|Inject epub.js| F[Render Book Chapter]

    F -->|User Swipes Page| G[Send 'Relocated' Event]
    G -->|PostMessage| H[Native Logic]
    H -->|Save CFI| I[AsyncStorage]

    F -->|User Clicks Chapter 2| J[Check Paywall State]
    J -->|Not Paid| K[Show Native Paywall Modal]
    J -->|Paid| L[Inject 'display(chapter2)']
```

---

3. Implementation Details

Phase A: The File Service (Utility Layer)

Create a helper to handle the `.epub` file. EPUBs are just ZIP files. Browsers cannot read ZIPs directly, so we must extract them.

File: `src/services/BookService.ts`

TypeScript

```
import RNFS from '@dr.pogodin/react-native-fs';
import { unzip } from 'react-native-zip-archive';

export const prepareBook = async (bookFileName: string): Promise<string> =>
{
  // 1. Define Paths
  // Source: Where the bundled epub lives (or downloaded location)
  const sourcePath = `${RNFS.DocumentDirectoryPath}/${bookFileName}.epub`;
  // Destination: Where we unzip it
  const targetPath = `${RNFS.DocumentDirectoryPath}/books/${bookFileName}`;

  // 2. Check if already unzipped
  const exists = await RNFS.exists(targetPath);
  if (exists) {
    return `file://${targetPath}`;
  }

  // 3. Unzip if needed
  try {
    // Ensure directory exists
    await RNFS.mkdir(`${RNFS.DocumentDirectoryPath}/books`);

    // Unzip
    await unzip(sourcePath, targetPath);
    return `file://${targetPath}`;
  } catch (error) {
    console.error('Failed to unzip book:', error);
    throw error;
  }
};
```

Phase B: The Reader Component (Web Layer)

This is the core component. It wraps the WebView and acts as the bridge.

File: `src/screens/Reader/ReaderScreen.tsx`

TypeScript

```
import React, { useEffect, useRef, useState } from 'react';
import { View, ActivityIndicator, StyleSheet, SafeAreaView } from 'react-
native';
import { WebView } from 'react-native-webview';
import { useRoute, useNavigation } from '@react-navigation/native';
import AsyncStorage from '@react-native-async-storage/async-storage';
import { prepareBook } from '../../services/BookService';
import { useSelector } from 'react-redux'; // Assuming you have user
selector
import TOCModal from './components/TOCModal'; // Native UI for Chapters

const ReaderScreen = () => {
  const route = useRoute();
  const navigation = useNavigation();
  const { bookId, fileName } = route.params as { bookId: string, fileName:
string };

  // State
  const [bookUrl, setBookUrl] = useState<string | null>(null);
  const [toc, setToc] = useState<any[]>([]);
  const [currentCfi, setCurrentCfi] = useState<string>('');
  const [isTocVisible, setTocVisible] = useState(false);

  // Refs
  const webViewRef = useRef<WebView>(null);

  // Selectors (Update based on your Redux slice)
  // const isPremium = useSelector(state => state.user.isPremium);
  const isPremium = false; // Hardcoded for testing

  // 1. Initialize Book
  useEffect(() => {
    const init = async () => {
      const url = await prepareBook(fileName);
      const lastLocation = await
AsyncStorage.getItem(`last_read_${bookId}`);
      setBookUrl(url);
      if (lastLocation) setCurrentCfi(lastLocation);
    };
    init();
  }, [bookId, fileName]);

  // 2. The HTML Template that runs inside WebView
  // We inject the epub.js library (CDN or local file)
  const renderHTML = () => `
    <!DOCTYPE html>
    <html>
    <head>
      <meta name="viewport" content="width=device-width, initial-scale=1.0,
maximum-scale=1.0, user-scalable=no">
      <script
src="https://cdnjs.cloudflare.com/ajax/libs/epub.js/0.3.93/epub.min.js"></s
cript>
      <style>
```

```
        body { margin: 0; height: 100vh; overflow: hidden; background:
#fff; }
      </style>
    </head>
    <body>
      <div id="viewer"></div>
      <script>
        // Initialize ePub
        // bookUrl will be injected by React Native
        var book = ePub("${bookUrl}/");
        var rendition = book.renderTo("viewer", {
          width: "100%",
          height: "100%",
          flow: "paginated",
          manager: "default"
        });

        // Display initial location
        var initialCfi = "${currentCfi}";
        initialCfi ? rendition.display(initialCfi) : rendition.display();

        // Event: Location Changed
        rendition.on("relocated", function(location){
          window.ReactNativeWebView.postMessage(JSON.stringify({
            type: 'LOC',
            cfi: location.start.cfi,
            percentage: location.start.percentage
          }));
        });

        // Event: Book Loaded (Get TOC)
        book.loaded.navigation.then(function(nav){
          window.ReactNativeWebView.postMessage(JSON.stringify({
            type: 'TOC',
            toc: nav.toc
          }));
        });
      </script>
    </body>
    </html>
  `;

  // 3. Handle Messages from WebView
  const onMessage = (event: any) => {
    try {
      const data = JSON.parse(event.nativeEvent.data);
      if (data.type === 'LOC') {
        AsyncStorage.setItem(`last_read_${bookId}`, data.cfi);
        setCurrentCfi(data.cfi);
      } else if (data.type === 'TOC') {
        setToc(data.toc);
      }
    } catch (e) {
      console.error(e);
    }
  };

  // 4. Navigation Logic (The Paywall Check)
  const handleChapterSelect = (chapterHref: string, index: number) => {
    // Logic: Chapter 0 is free. Chapter 1+ requires Premium.
    if (index > 0 && !isPremium) {
```

```
      setTocVisible(false);
      // Navigate to your Paywall Screen
      // navigation.navigate('Paywall');
      alert("Please upgrade to read this chapter.");
      return;
    }

    // If allowed, tell WebView to render that chapter
    webViewRef.current?.injectJavaScript(`
      rendition.display('${chapterHref}');
      true;
    `);
    setTocVisible(false);
  };

  if (!bookUrl) return <ActivityIndicator size="large" style={{flex:1}} />;

  return (
    <SafeAreaView style={styles.container}>
      <WebView
        ref={webViewRef}
        originWhitelist={['*']}
        source={{ html: renderHTML(), baseUrl: bookUrl }}
        onMessage={onMessage}
        allowFileAccess={true}
        allowUniversalAccessFromFileURLs={true} // Essential for Android to
read local EPUB files
        allowingReadAccessToURL={bookUrl} // Essential for iOS
        style={{ flex: 1 }}
      />

      {/* Custom Native UI Overlay for Menu */}
      <TOCModal
        visible={isTocVisible}
        toc={toc}
        onClose={() => setTocVisible(false)}
        onSelect={handleChapterSelect}
        isPremium={isPremium}
      />
    </SafeAreaView>
  );
};

const styles = StyleSheet.create({
  container: { flex: 1, backgroundColor: '#fff' },
});

export default ReaderScreen;
```

Phase C: The Native UI (TOC & Paywall)

We render the Table of Contents in Native code (using `react-native-vector-icons` and `react-native-paper`) to ensure it feels smooth and responsive, unlike a web-based menu.

File: `src/screens/Reader/components/TOCModal.tsx`

TypeScript
```
import React from 'react';
```

```
import { Modal, View, FlatList, TouchableOpacity, StyleSheet } from 'react-
native';
import { Text, IconButton, useTheme } from 'react-native-paper';
import Icon from 'react-native-vector-icons/MaterialCommunityIcons';

interface TOCProps {
  visible: boolean;
  toc: any[];
  onClose: () => void;
  onSelect: (href: string, index: number) => void;
  isPremium: boolean;
}

const TOCModal = ({ visible, toc, onClose, onSelect, isPremium }: TOCProps)
=> {
  const theme = useTheme();

  const renderItem = ({ item, index }: { item: any, index: number }) => {
    const isLocked = index > 0 && !isPremium; // First chapter free

    return (
      <TouchableOpacity
        style={[styles.item, { borderBottomColor: theme.colors.outline }]}
        onPress={() => onSelect(item.href, index)}
      >
        <Text style={[styles.label, isLocked && styles.lockedText]}>
            {item.label.trim()}
        </Text>
        {isLocked && (
          <Icon name="lock" size={20} color={theme.colors.error} />
        )}
      </TouchableOpacity>
    );
  };

  return (
    <Modal visible={visible} animationType="slide"
presentationStyle="pageSheet">
      <View style={styles.container}>
        <View style={styles.header}>
          <Text variant="titleLarge">Table of Contents</Text>
          <IconButton icon="close" onPress={onClose} />
        </View>
        <FlatList
          data={toc}
          keyExtractor={(item) => item.id || item.href}
          renderItem={renderItem}
          contentContainerStyle={styles.list}
        />
      </View>
    </Modal>
  );
};

const styles = StyleSheet.create({
  container: { flex: 1, backgroundColor: '#fff' },
  header: { flexDirection: 'row', justifyContent: 'space-between',
alignItems: 'center', padding: 16 },
  list: { paddingHorizontal: 16 },
  item: { flexDirection: 'row', justifyContent: 'space-between',
paddingVertical: 16, borderBottomWidth: 1 },
```

```
  label: { fontSize: 16 },
  lockedText: { opacity: 0.5 }
});

export default TOCModal;
```

---

4. Important Configuration Notes

1. Android File Access

In `android/app/src/main/AndroidManifest.xml`, ensure you have permission to read storage (though scoped storage usually handles app-specific files fine):

XML
```
<uses-permission android:name="android.permission.READ_EXTERNAL_STORAGE" />
```

Note: Since we are writing to `DocumentDirectoryPath` (internal app storage), explicit permissions are often not needed for standard extraction, but `allowUniversalAccessFromFileURLs={true}` in the WebView is mandatory.

2. iOS File Access

In your code, the `allowingReadAccessToURL={bookUrl}` prop in the WebView is critical. Without it, the WebView cannot load the CSS or Images extracted from the EPUB.

3. Styling Strategy

Because we are using `epub.js` inside a WebView, **you do not need to write CSS**. The library will render the book exactly as the `.epub` file defines it (which you mentioned is already perfect).

4. Why not `@epubjs-react-native/core`?

While you have this in your `package.json`, this library wraps the logic above. However, for **Paywall integration**, it is safer to manage the WebView manually as shown above. It gives you direct access to the `onChapterSelect` logic where you can inject the "Is User Paid?" check before the render happens.