Problem Set I - Regex

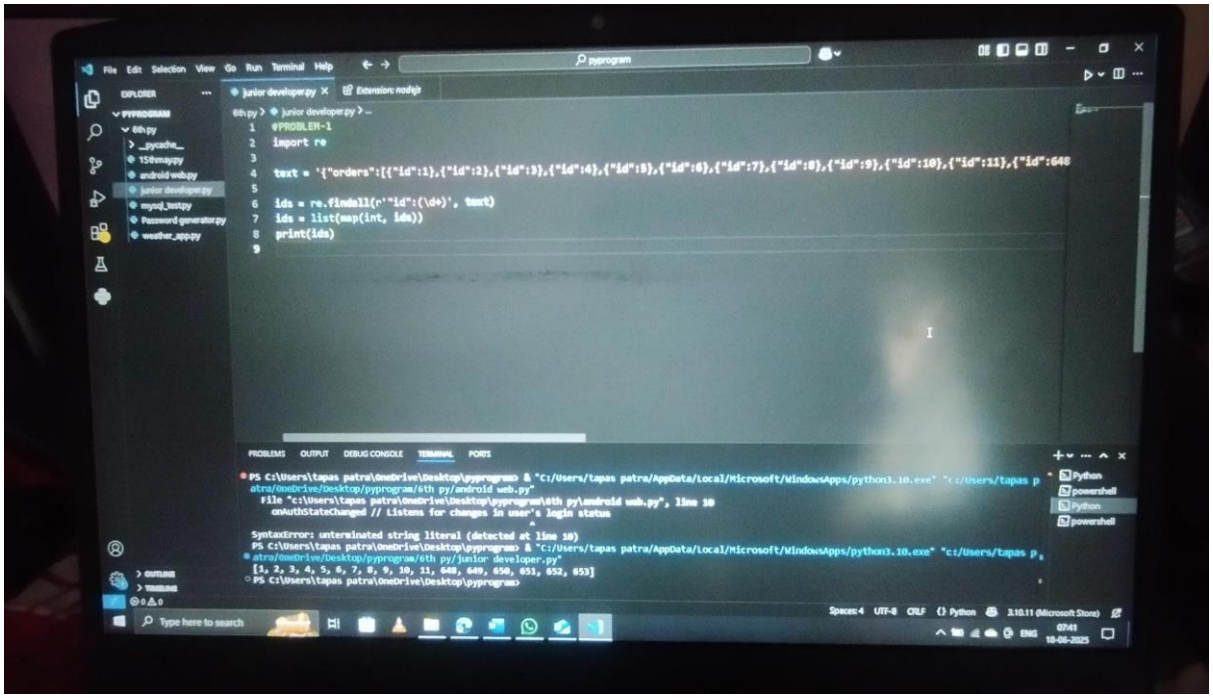
1.Write a regex to extract all the numbers with orange color background from the below text in italics (Output should be a list).

Solution:-

```
import re


text =
'{"orders":[{"id":1},{"id":2},{"id":3},{"id":4},{"id":5},{"id":6},{"id":7},{"id":8},{"id":9},{"id":10},{"id":11},{"id":648},{"id":649},{"id":650},{"id":651},{"id":652},{"id":653}],"errors":[{"code":3,"message":"[PHP Warning #2] count(): Parameter must be an array or an object that implements Countable (153)"}]}'


ids = re.findall(r'"id":(\d+)', text)

ids = list(map(int, ids))

print(ids)
```

O/P:-

[1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 648, 649, 650, 651, 652, 653]



Problem Set 2 - A functioning web app with API

1. **Admin Facing** - Where admin user can add an **android app** as well as the number of points - earned by user for downloading the app. (Please do not use the default Django Admin)
2. **User Facing** - where the user can see the apps added by the admin and the points. The user should be able to see the following fields.

- Signup and Login (Feel free to use any package for the same).
- Their Name and Profile
- Points Earned.

- Tasks completed.
- Option to upload a screenshot (which must include drag and drop) for that particular task. (Like if a user downloads a particular app), he can send a screenshot of the open app to confirm that he has indeed downloaded the app.

Solution:-

Code:-

```
import React, { useState, useEffect, useContext, createContext } from 'react';
import { initializeApp } from 'firebase/app';
import {
  getAuth,
  signInWithCustomToken,
  signInAnonymously,
  createUserWithEmailAndPassword,
  signInWithEmailAndPassword,
  signOut,
  onAuthStateChanged // Listens for changes in user's login status
} from 'firebase/auth';
import {
  getFirestore,
```

```
  doc,
  setDoc,      // To create or overwrite a document
  getDoc,      // To read a single document
  collection,  // To get a reference to a collection
  query,       // To build complex queries (like filtering)
  onSnapshot,  // To listen for real-time updates to data
  addDoc,      // To add a new document with an auto-generated ID
  updateDoc,   // To update specific fields in a document
  arrayUnion   // To add elements to an array field without duplicates
} from 'firebase/firestore';
import { getStorage, ref, uploadBytes, getDownloadURL } from 'firebase/storage';

// --- IMPORTANT GLOBAL VARIABLES ---
// These variables are provided by the environment where this code runs (like Canvas).
// We check if they exist; if not, we use a default value to prevent errors.
const appId = typeof __app_id !== 'undefined' ? __app_id : 'default-app-id';
const firebaseConfig = typeof __firebase_config !== 'undefined' ? JSON.parse(__firebase_config) : {};
```

```
const initialAuthToken = typeof __initial_auth_token !== 'undefined' ? __initial_auth_token : null;

// --- INITIALIZE FIREBASE SERVICES ---
// We set up our Firebase connections here. Think of this as getting our tools ready.
const firebaseApp = initializeApp(firebaseConfig); // The main Firebase app instance
const auth = getAuth(firebaseApp); // Service for user authentication (login/signup)
const db = getFirestore(firebaseApp); // Service for our database (Firestore)
const storage = getStorage(firebaseApp); // Service for storing files (like images)

// --- CONTEXTS FOR SHARING DATA ---
// Contexts are like special "global" pockets where we can store information
// and share it with many components without passing it manually everywhere.

// AuthContext: Stores user login status, profile, and Firebase service instances.
const AuthContext = createContext(null);
```

```
// PageContext: Stores which page the user is currently viewing.
const PageContext = createContext(null);


// --- AUTH PROVIDER COMPONENT ---
// This component manages all authentication logic and user profile loading.
// It wraps around our entire application so any component can access auth info.
const AuthProvider = ({ children }) => {
  // State variables to hold authentication and user profile data
  const [currentUser, setCurrentUser] = useState(null); // The Firebase user object
  const [loadingAuth, setLoadingAuth] = useState(true); // True while we're checking login status
  const [currentUserId, setCurrentUserId] = useState(null); // The unique ID of the logged-in user
  const [userRole, setUserRole] = useState(null); // 'admin' or 'user'
  const [userProfile, setUserProfile] = useState(null); // Full user data (name, points, etc.)
```

```
  // useEffect: Runs code when the component first loads or when dependencies change.
  // The empty array `[]` means it runs only once, when the component mounts.
  useEffect(() => {
    // Function to sign in the user initially
    const performInitialSignIn = async () => {
      try {
        if (initialAuthToken) {
          // If we have an initial token, sign in with it (for Canvas environment)
          await signInWithCustomToken(auth, initialAuthToken);
        } else {
          // Otherwise, sign in anonymously (user not explicitly logged in)
          await signInAnonymously(auth);
        }
      } catch (error) {
        console.error("Error during initial Firebase sign-in:", error);
      } finally {
        setLoadingAuth(false); // Authentication check is complete
```

```
    }
  };

  performInitialSignIn(); // Call the initial sign-in function

  // onAuthStateChanged: This is a Firebase listener. It runs whenever the user's
  // login status changes (e.g., logs in, logs out, account created).
  const unsubscribe = onAuthStateChanged(auth, async (user) => {
    setCurrentUser(user); // Update the React state with the current user object
    if (user) {
      // If a user is logged in (or anonymous), get their unique ID
      const userIdFromAuth = user.uid;
      setCurrentUserId(userIdFromAuth);

      // Try to fetch the user's profile from Firestore
      // Path: artifacts/{appId}/users/{userId}/profile/data
      const userProfileDocRef = doc(db, `artifacts/${appId}/users/${userIdFromAuth}/profile`, 'data');
```

```
      const userProfileDocSnap = await
getDoc(userProfileDocRef);


      if (userProfileDocSnap.exists()) {
        // If profile exists, load its data
        const userData = userProfileDocSnap.data();
        setUserProfile(userData);
        setUserRole(userData.role || 'user'); // Set role, default
to 'user'
        console.log(`User profile loaded: ${userData.name ||
'N/A'}, Role: ${userData.role || 'user'}`);
      } else {
        // If profile doesn't exist (e.g., brand new signup), create
a default one
        console.log("User profile does not exist, creating new
default profile.");
        const defaultProfile = {
          name: user.email ? user.email.split('@')[0] : `user-
${userIdFromAuth.substring(0, 6)}`,
          email: user.email || 'anonymous@example.com',
          role: 'user', // Default role for all new users
          points: 0,
          completedTasks: []
```

```
          };
          await setDoc(userProfileDocRef, defaultProfile); // Save the default profile
          setUserProfile(defaultProfile);
          setUserRole('user');
        }
      } else {
        // If no user is logged in, clear all user-related state
        setCurrentUserId(null);
        setUserRole(null);
        setUserProfile(null);
      }
      setLoadingAuth(false); // Authentication status has been fully checked
    });


    // Cleanup function: This runs when the AuthProvider component is removed.
    // It stops listening for auth state changes to prevent memory leaks.
    return () => unsubscribe();
  }, []); // Empty dependency array: runs only once on component mount
```

```
  // The 'value' object contains all the data and functions we want to share
  // with components that use AuthContext.
  const authContextValue = {
    currentUser,
    currentUserId, // Renamed from userId for clarity
    userRole,
    userProfile,
    loadingAuth,
    db, // Firestore database instance
    auth, // Firebase Auth instance
    storage, // Firebase Storage instance
    appId, // The application ID
    updateUserProfile: setUserProfile, // Function to update user profile in local state
  };


  return (
    // AuthContext.Provider makes the authContextValue available to its children.
    // We only render children once authentication loading is complete.
```

```
    <AuthContext.Provider value={authContextValue}>
      {!loadingAuth && children}
    </AuthContext.Provider>
  );
};

// --- PAGE PROVIDER COMPONENT ---
// This manages which main page is currently displayed (Login, Admin Panel, User Dashboard, etc.).
const PageProvider = ({ children }) => {
  // `currentPage` state tracks the name of the currently active page.
  const [currentPage, setCurrentPage] = useState('login'); // Start at the login page by default

  // The value provided by this context includes the current page and a way to change it.
  const pageContextValue = { currentPage, setCurrentPage };

  return (
    <PageContext.Provider value={pageContextValue}>
      {children}
    </PageContext.Provider>
```

```
  );
};

// --- HELPER UI COMPONENTS ---
// These are small, reusable components for common UI elements.

// Header: The top navigation bar.
const Header = () => {
  // Get user info and role from AuthContext.
  const { currentUser, userRole, userProfile } = useContext(AuthContext);
  // Get function to change page from PageContext.
  const { setCurrentPage } = useContext(PageContext);

  // handleLogout: Function to sign out the current user.
  const handleLogout = async () => {
    try {
      await signOut(auth); // Firebase Auth signs out the user.
      window.location.reload(); // Reloads the page to fully reset the app state.
    } catch (error) {
      console.error("Error signing out:", error);
```

```
  }
 };


 return (
  <header className="bg-gradient-to-r from-purple-600 to-indigo-700 text-white p-4 shadow-lg rounded-b-lg">
    <div className="container mx-auto flex justify-between items-center">
      <h1 className="text-3xl font-bold font-inter">App Reward System</h1>
      <nav>
       {currentUser ? (
        // If a user is logged in, show their name, role, and a Logout button.
        <div className="flex items-center space-x-4">
         <span className="text-lg">
          Hello, {userProfile?.name || 'Guest'}! ({userRole === 'admin' ? 'Admin' : 'User'})
         </span>
         <button
          onClick={handleLogout}
          className="bg-white text-purple-700 font-semibold py-2 px-4 rounded-lg shadow-md hover:bg-gray-100 transition duration-300"
```

```
            >
              Logout
            </button>
          </div>
        ) : (
          // If no user is logged in, show Login and Signup buttons.
          <div className="flex space-x-4">
            <LinkButton text="Login" target="login" />
            <LinkButton text="Signup" target="signup" />
          </div>
        )}
      </nav>
    </div>
  </header>
 );
};


// LinkButton: A reusable button that changes the current page.
const LinkButton = ({ text, target, onClick }) => {
  const { setCurrentPage } = useContext(PageContext); // Get function to change page
```

```
  return (
    <button
      onClick={() => {
        if (onClick) onClick(); // If there's an extra click handler, run it.
        setCurrentPage(target); // Change the current page state.
      }}
      className="bg-purple-700 hover:bg-purple-800 text-white font-semibold py-2 px-4 rounded-lg shadow-md transition duration-300 transform hover:scale-105"
    >
      {text}
    </button>
  );
};

// LoadingSpinner: Shows a spinning animation while data is loading.
const LoadingSpinner = () => (
  <div className="flex justify-center items-center h-screen">
    <div className="animate-spin rounded-full h-16 w-16 border-t-4 border-b-4 border-purple-500"></div>
```

```
    <p className="ml-4 text-lg text-gray-700">Loading...</p>
  </div>
);

// MessageModal: A custom pop-up box to display messages (instead of alert()).
const MessageModal = ({ message, onClose }) => {
  if (!message) return null; // Don't show modal if there's no message.
  return (
    <div className="fixed inset-0 bg-black bg-opacity-50 flex items-center justify-center z-50">
      <div className="bg-white p-6 rounded-lg shadow-xl max-w-sm w-full text-center">
        <p className="text-lg mb-4">{message}</p>
        <button
          onClick={onClose}
          className="bg-purple-600 text-white py-2 px-4 rounded-md hover:bg-purple-700 transition duration-300"
        >
          Close
        </button>
      </div>
```

```
    </div>
  );
};

// --- AUTHENTICATION PAGES (Login/Signup) ---

// AuthForm: Handles both Signup and Login forms based on the 'type' prop.
const AuthForm = ({ type }) => {
  // State for form inputs
  const [email, setEmail] = useState('');
  const [password, setPassword] = useState('');
  const [name, setName] = useState(''); // Only used for 'signup'
  // State for messages and modal
  const [message, setMessage] = useState('');
  const [isModalVisible, setIsModalVisible] = useState(false);

  // Get functions/objects from contexts
  const { setCurrentPage } = useContext(PageContext);
  const { db, auth, appId } = useContext(AuthContext);
```

```
  // handleSubmit: Handles form submission for login or signup.
  const handleSubmit = async (event) => {
    event.preventDefault(); // Stop the browser from reloading the page
    setMessage(''); // Clear previous messages

    try {
      if (type === 'signup') {
        // Create a new user account with email and password
        const userCredential = await createUserWithEmailAndPassword(auth, email, password);
        const user = userCredential.user;

        // Save new user's profile data to Firestore
        // Path: artifacts/{appId}/users/{userId}/profile/data
        const userProfileDocRef = doc(db, `artifacts/${appId}/users/${user.uid}/profile`, 'data');
        await setDoc(userProfileDocRef, {
          name: name,
          email: email,
          role: 'user', // All new signups are 'user' by default
          points: 0,
```

```
        completedTasks: []
      });

      setMessage('Signup successful! You can now log in.');
      setIsModalVisible(true);
      // Clear form fields
      setEmail('');
      setPassword('');
      setName('');
      setCurrentPage('login'); // After signup, go to login page
    } else { // type === 'login'
      // Sign in an existing user with email and password
      await signInWithEmailAndPassword(auth, email, password);
      setMessage('Login successful!');
      setIsModalVisible(true);
      setCurrentPage('dashboard'); // After login, go to the dashboard
    }
  } catch (error) {
    console.error("Authentication error:", error);
    let errorMessage = "An unknown error occurred during authentication.";
```

```
      // Provide user-friendly error messages based on Firebase error codes
      if (error.code === 'auth/email-already-in-use') {
        errorMessage = 'This email is already registered. Please login or use a different email.';
      } else if (error.code === 'auth/invalid-email') {
        errorMessage = 'The email address is not valid.';
      } else if (error.code === 'auth/weak-password') {
        errorMessage = 'The password is too weak. Please use at least 6 characters.';
      } else if (error.code === 'auth/user-not-found' || error.code === 'auth/wrong-password') {
        errorMessage = 'Invalid email or password. Please try again.';
      }
      setMessage(errorMessage);
      setIsModalVisible(true);
    }
  };

  // handleModalClose: Function to close the message modal.
  const handleModalClose = () => {
    setIsModalVisible(false);
```

```
    setMessage('');
  };


  return (
    // The auth forms will now fill the available height in the main flex container
    <div className="flex justify-center items-center h-full bg-gray-100 py-10 px-4">
      <div className="bg-white p-8 rounded-xl shadow-2xl w-full max-w-md border border-gray-200">
        <h2 className="text-3xl font-extrabold text-center text-purple-700 mb-8">
          {type === 'signup' ? 'Create New Account' : 'Welcome Back! Login'}
        </h2>
        <form onSubmit={handleSubmit} className="space-y-6">
          {type === 'signup' && (
            <div>
              <label htmlFor="name-input" className="block text-gray-700 text-sm font-medium mb-2">
                Your Name
              </label>
              <input
```

```
            type="text"
            id="name-input"
            value={name}
            onChange={(e) => setName(e.target.value)}
            required
            className="w-full px-4 py-2 border border-gray-300 rounded-lg focus:outline-none focus:ring-2 focus:ring-purple-500 transition duration-200"
            placeholder="e.g., Jane Doe"
          />
        </div>
      )}
      <div>
        <label htmlFor="email-input" className="block text-gray-700 text-sm font-medium mb-2">
          Email Address
        </label>
        <input
          type="email"
          id="email-input"
          value={email}
          onChange={(e) => setEmail(e.target.value)}
          required
```

```
          className="w-full px-4 py-2 border border-gray-300 rounded-lg focus:outline-none focus:ring-2 focus:ring-purple-500 transition duration-200"
          placeholder="e.g., your_email@example.com"
        />
      </div>
      <div>
        <label htmlFor="password-input" className="block text-gray-700 text-sm font-medium mb-2">
          Password
        </label>
        <input
          type="password"
          id="password-input"
          value={password}
          onChange={(e) => setPassword(e.target.value)}
          required
          className="w-full px-4 py-2 border border-gray-300 rounded-lg focus:outline-none focus:ring-2 focus:ring-purple-500 transition duration-200"
          placeholder="Minimum 6 characters"
        />
      </div>
```

```
        <button
          type="submit"
          className="w-full bg-purple-600 text-white font-semibold py-3 px-4 rounded-lg shadow-lg hover:bg-purple-700 transition duration-300 transform hover:scale-105"
        >
          {type === 'signup' ? 'Sign Up Now' : 'Log In Securely'}
        </button>
      </form>
      <div className="mt-6 text-center text-gray-600">
        {type === 'signup' ? (
          <>
            Already have an account?{' '}
            <button
              onClick={() => setCurrentPage('login')}
              className="text-purple-600 hover:text-purple-800 font-medium hover:underline"
            >
              Log In Here
            </button>
          </>
        ) : (
          <>
```

```
            Don't have an account yet?{' '}
            <button
              onClick={() => setCurrentPage('signup')}
              className="text-purple-600 hover:text-purple-800
font-medium hover:underline"
            >
              Sign Up Now
            </button>
          </>
        )}
      </div>
    </div>
    <MessageModal message={message}
onClose={handleModalClose} />
  </div>
 );
};

// --- ADMIN PANEL COMPONENTS ---
// These components are only visible to users with 'admin'
role.

// AdminPanel: The main dashboard for administrators.
```

```
const AdminPanel = () => {
  const { setCurrentPage } = useContext(PageContext);
  const { currentUserId } = useContext(AuthContext); // Display admin's user ID

  return (
    <div className="container mx-auto p-6 bg-white rounded-xl shadow-lg mt-8">
      <h2 className="text-4xl font-extrabold text-purple-800 mb-6 text-center">Admin Dashboard</h2>
      <p className="text-center text-gray-600 mb-6">Your Admin User ID: <span className="font-mono text-purple-700 break-words">{currentUserId}</span></p>

      <div className="grid grid-cols-1 md:grid-cols-2 gap-6 mb-8">
        <button
          onClick={() => setCurrentPage('admin-add-app')}
          className="flex items-center justify-center bg-gradient-to-r from-green-500 to-teal-600 text-white font-semibold py-4 px-6 rounded-lg shadow-xl hover:shadow-2xl transition duration-300 transform hover:scale-105 text-lg"
        >
          {/* SVG icon for Add New App */}
```

<svg xmlns="http://www.w3.org/2000/svg" width="24" height="24" viewBox="0 0 24 24" fill="none" stroke="currentColor" strokeWidth="2" strokeLinecap="round" strokeLinejoin="round" className="mr-3"><path d="M12 5v14M5 12h14"/></svg>

Add New App

</button>

<button

onClick={() => setCurrentPage('admin-view-apps')}

className="flex items-center justify-center bg-gradient-to-r from-blue-500 to-indigo-600 text-white font-semibold py-4 px-6 rounded-lg shadow-xl hover:shadow-2xl transition duration-300 transform hover:scale-105 text-lg"

>

{/* SVG icon for View All Apps */}

<svg xmlns="http://www.w3.org/2000/svg" width="24" height="24" viewBox="0 0 24 24" fill="none" stroke="currentColor" strokeWidth="2" strokeLinecap="round" strokeLinejoin="round" className="mr-3"><path d="M22 12H4M2 9l2-3.5L7 9M22 9l-2-3.5L17 9"/></svg>

View All Apps

</button>

```
    <div className="mt-8">
      <h3 className="text-2xl font-bold text-gray-800 mb-4 text-center">Important: How to Make a User an Admin</h3>
      <p className="text-gray-700 text-center leading-relaxed">
        After a user signs up (for example, with an email like `admin@yourdomain.com`), you need to
        manually change their `role` in your Firebase Firestore database.
        Find their user document at this path:
      </p>
      <code className="block bg-gray-100 p-3 rounded-md my-4 font-mono text-sm text-gray-800 break-words">
        artifacts/{appId}/users/{'{the_user_ID}'}/profile/data
      </code>
      <p className="text-gray-700 text-center leading-relaxed">
        Change the value of the `role` field from `user` to `admin`. Once you do this,
        that user will see this Admin Dashboard when they log in.
      </p>
    </div>
  </div>
```

```
  );
};

// AdminAddApp: Form for administrators to add new Android applications.
const AdminAddApp = () => {
  // State variables for form inputs
  const [appName, setAppName] = useState('');
  const [appUrl, setAppUrl] = useState('');
  const [pointsAwarded, setPointsAwarded] = useState(''); // Renamed from 'points' for clarity
  const [appDescription, setAppDescription] = useState(''); // Renamed from 'description' for clarity
  // State for messages and modal
  const [message, setMessage] = useState('');
  const [isModalVisible, setIsModalVisible] = useState(false);

  // Get functions/objects from contexts
  const { setCurrentPage } = useContext(PageContext);
  const { db, appId } = useContext(AuthContext);

  // handleSubmit: Function to save a new app to Firestore.
  const handleSubmit = async (event) => {
```

```
    event.preventDefault();
    setMessage(''); // Clear previous messages

    // Basic validation for form fields
    if (!appName || !appUrl || pointsAwarded === '' ||
!appDescription) {
      setMessage('All fields are required to add an app.');
      setIsModalVisible(true);
      return;
    }
    if (isNaN(parseInt(pointsAwarded)) ||
parseInt(pointsAwarded) <= 0) {
      setMessage('Points must be a positive number.');
      setIsModalVisible(true);
      return;
    }

    try {
      // Add a new document to the 'apps' collection.
      // This data is public, so it's stored under:
artifacts/{appId}/public/data/apps
      await addDoc(collection(db,
`artifacts/${appId}/public/data/apps`), {
```

```
      appName: appName,
      appUrl: appUrl,
      points: parseInt(pointsAwarded), // Convert points to a
number
      description: appDescription,
      createdAt: new Date(), // Record when the app was
added
    });

    setMessage('App successfully added!');
    setIsModalVisible(true);
    // Clear form fields after successful submission
    setAppName('');
    setAppUrl('');
    setPointsAwarded('');
    setAppDescription('');
  } catch (error) {
    console.error("Error adding app:", error);
    setMessage('Failed to add app. Please try again.');
    setIsModalVisible(true);
  }
 };
```

```
  // handleModalClose: Function to close the message modal.
  const handleModalClose = () => {
    setIsModalVisible(false);
    setMessage('');
  };

  return (
    <div className="container mx-auto p-6 bg-white rounded-xl shadow-lg mt-8">
      <h2 className="text-3xl font-extrabold text-purple-700 mb-6 text-center">Add New Android App</h2>
      <form onSubmit={handleSubmit} className="space-y-6">
        <div>
          <label htmlFor="app-name-input" className="block text-gray-700 text-sm font-medium mb-2">
            App Name
          </label>
          <input
            type="text"
            id="app-name-input"
            value={appName}
            onChange={(e) => setAppName(e.target.value)}
```

```
          required
          className="w-full px-4 py-2 border border-gray-300 rounded-lg focus:outline-none focus:ring-2 focus:ring-purple-500"
          placeholder="e.g., My New Exciting App"
        />
      </div>
      <div>
        <label htmlFor="app-url-input" className="block text-gray-700 text-sm font-medium mb-2">
          App Download URL (e.g., Google Play Store Link)
        </label>
        <input
          type="url"
          id="app-url-input"
          value={appUrl}
          onChange={(e) => setAppUrl(e.target.value)}
          required
          className="w-full px-4 py-2 border border-gray-300 rounded-lg focus:outline-none focus:ring-2 focus:ring-purple-500"
          placeholder="https://play.google.com/store/apps/details?id=..."
```

```
        />
      </div>
      <div>
        <label htmlFor="points-input" className="block text-gray-700 text-sm font-medium mb-2">
          Points Awarded for Download
        </label>
        <input
          type="number"
          id="points-input"
          value={pointsAwarded}
          onChange={(e) => setPointsAwarded(e.target.value)}
          required
          min="1" // Ensure at least 1 point is awarded
          className="w-full px-4 py-2 border border-gray-300 rounded-lg focus:outline-none focus:ring-2 focus:ring-purple-500"
          placeholder="e.g., 100"
        />
      </div>
      <div>
        <label htmlFor="description-textarea" className="block text-gray-700 text-sm font-medium mb-2">
```

```
          App Description
        </label>
        <textarea
          id="description-textarea"
          value={appDescription}
          onChange={(e) => setAppDescription(e.target.value)}
          required
          rows="4" // Set initial height for textarea
          className="w-full px-4 py-2 border border-gray-300
rounded-lg focus:outline-none focus:ring-2 focus:ring-purple-
500 resize-y"
          placeholder="Write a brief description of the app and
how users earn points."
        ></textarea>
      </div>
      <button
        type="submit"
        className="w-full bg-purple-600 text-white font-
semibold py-3 px-4 rounded-lg shadow-lg hover:bg-purple-
700 transition duration-300 transform hover:scale-105"
      >
        Add App to System
      </button>
```

```
      </form>
      <div className="mt-6 text-center">
        <button
          onClick={() => setCurrentPage('admin-panel')}
          className="text-purple-600 hover:text-purple-800 font-medium hover:underline"
        >
          Back to Admin Dashboard
        </button>
      </div>
      <MessageModal message={message} onClose={handleModalClose} />
    </div>
  );
};

// AdminViewApps: Displays a list of all apps added by administrators.
const AdminViewApps = () => {
  const [appList, setAppList] = useState([]); // State to hold the list of apps
  const [isLoadingApps, setIsLoadingApps] = useState(true); // State to show loading spinner
```

```
  // Get objects from contexts
  const { db, appId } = useContext(AuthContext);
  const { setCurrentPage } = useContext(PageContext);

  useEffect(() => {
    // Reference to the 'apps' collection in Firestore.
    // Path: artifacts/{appId}/public/data/apps
    const appsCollectionRef = collection(db,
`artifacts/${appId}/public/data/apps`);

    // onSnapshot: Listens for real-time changes to the 'apps'
collection.
    // Whenever an app is added, changed, or deleted, this
function runs.
    const unsubscribe = onSnapshot(appsCollectionRef,
(snapshot) => {
      const fetchedApps = snapshot.docs.map(docData => ({
        id: docData.id, // Get the unique ID of the document
        ...docData.data() // Get all other data fields
      }));
      setAppList(fetchedApps);
      setIsLoadingApps(false); // Done loading
```

```
    }, (error) => {
      console.error("Error fetching apps in AdminViewApps:", error);
      setIsLoadingApps(false); // Stop loading even if there's an error
    });

    // Cleanup function: Stops listening for updates when the component unmounts.
    return () => unsubscribe();
  }, [db, appId]); // Dependencies: Re-run if db or appId changes (unlikely for these constants)

  if (isLoadingApps) {
    return <LoadingSpinner />; // Show spinner while loading apps
  }

  return (
    <div className="container mx-auto p-6 bg-white rounded-xl shadow-lg mt-8">
      <h2 className="text-3xl font-extrabold text-purple-700 mb-6 text-center">All Available Apps</h2>
      {appList.length === 0 ? (
```

```
      <p className="text-center text-gray-600 text-lg">No apps have been added yet by an admin.</p>
    ) : (
      <div className="grid grid-cols-1 md:grid-cols-2 lg:grid-cols-3 gap-6">
        {appList.map((app) => (
          <div key={app.id} className="bg-gray-50 p-6 rounded-lg shadow-md border border-gray-200">
            <h3 className="text-xl font-bold text-gray-800 mb-2">{app.appName}</h3>
            <p className="text-purple-600 font-semibold mb-3">{app.points} Points</p>
            <p className="text-gray-600 text-sm mb-4">{app.description}</p>
            <a
              href={app.appUrl}
              target="_blank" // Opens link in a new tab
              rel="noopener noreferrer" // Security best practice for target="_blank"
              className="inline-flex items-center text-blue-600 hover:text-blue-800 font-medium transition duration-200"
            >
              Download App
              {/* SVG icon for external link */}
```

```
                <svg xmlns="http://www.w3.org/2000/svg"
width="16" height="16" viewBox="0 0 24 24" fill="none"
stroke="currentColor" strokeWidth="2"
strokeLinecap="round" strokeLinejoin="round"
className="ml-1"><path d="M18 13v6a2 2 0 0 1-2 2H5a2 2
0 = 0 1-2-2V8a2 2 0 0 1 2-2h6"/><polyline points="15 3 21 3
21 9"/><line x1="10" y1="14" x2="21" y2="3"/></svg>
              </a>
            </div>
          ))}
        </div>
      )}
      <div className="mt-8 text-center">
        <button
          onClick={() => setCurrentPage('admin-panel')}
          className="text-purple-600 hover:text-purple-800
font-medium hover:underline"
        >
          Back to Admin Dashboard
        </button>
      </div>
    </div>
  );
};
```

```
// --- USER PANEL COMPONENTS ---
// These components are visible to regular users.

// UserPanel: The main dashboard for users, showing their profile and points.
const UserPanel = () => {
  const { setCurrentPage } = useContext(PageContext);
  const { userProfile, currentUserId } = useContext(AuthContext); // Get user profile and ID

  if (!userProfile) {
    return <LoadingSpinner />; // Show spinner if user profile is still loading
  }

  return (
    <div className="container mx-auto p-6 bg-white rounded-xl shadow-lg mt-8">
      <h2 className="text-4xl font-extrabold text-purple-800 mb-6 text-center">Your User Dashboard</h2>
      <p className="text-center text-gray-600 mb-6">Your Unique User ID: <span className="font-mono text-purple-700 break-words">{currentUserId}</span></p>
```

```
    <div className="grid grid-cols-1 md:grid-cols-2 gap-6 mb-8">
      {/* User Profile Card */}
      <div className="bg-gradient-to-r from-blue-100 to-blue-200 p-6 rounded-lg shadow-md border border-blue-300">
        <h3 className="text-2xl font-bold text-blue-700 mb-3">Your Profile Information</h3>
        <p className="text-gray-700 mb-2"><strong>Name:</strong> {userProfile.name}</p>
        <p className="text-gray-700 mb-2"><strong>Email:</strong> {userProfile.email}</p>
        <p className="text-gray-700 font-semibold text-lg">
          <strong>Total Points Earned:</strong> <span className="text-green-600">{userProfile.points}</span>
        </p>
      </div>
      {/* Completed Tasks Card */}
      <div className="bg-gradient-to-r from-green-100 to-green-200 p-6 rounded-lg shadow-md border border-green-300">
        <h3 className="text-2xl font-bold text-green-700 mb-3">Your Completed Tasks</h3>
```

```
        {userProfile.completedTasks &&
userProfile.completedTasks.length > 0 ? (
          <ul className="list-disc pl-5 text-gray-700">
            {userProfile.completedTasks.map((task, index) => (
              <li key={index} className="mb-1 text-base">
                {task.appName} - <span className="font-semibold
text-purple-600">{task.pointsEarned} Points</span>
({task.status})
              </li>
            ))}
          </ul>
        ) : (
          <p className="text-gray-600">You haven't completed
any tasks yet! Start earning points.</p>
        )}
      </div>
    </div>

    <div className="mt-8 text-center">
      <button
        onClick={() => setCurrentPage('user-tasks')}
```

```
        className="bg-purple-600 text-white font-semibold py-3 px-6 rounded-lg shadow-lg hover:bg-purple-700 transition duration-300 transform hover:scale-105 text-lg"
      >
        View Available Tasks
      </button>
    </div>
  </div>
 );
};

// UserTaskList: Displays all available apps (tasks) for users to complete.
const UserTaskList = () => {
 const [availableApps, setAvailableApps] = useState([]); // State for list of apps
 const [isLoadingTasks, setIsLoadingTasks] = useState(true); // State for loading spinner
 const [selectedAppForUpload, setSelectedAppForUpload] = useState(null); // App selected for screenshot upload
 // State for messages and modal
 const [message, setMessage] = useState('');
 const [isMessageModalVisible, setIsMessageModalVisible] = useState(false);
```

```
  // Get functions/objects from contexts
  const { db, appId, userProfile, updateUserProfile,
currentUserId } = useContext(AuthContext);
  const { setCurrentPage } = useContext(PageContext);

  // handleMessageModalClose: Function to close the general
message modal.
  const handleMessageModalClose = () => {
    setIsMessageModalVisible(false);
    setMessage('');
  };

  useEffect(() => {
    // Reference to the public 'apps' collection.
    const appsCollectionRef = collection(db,
`artifacts/${appId}/public/data/apps`);

    // Listen for real-time updates to available apps.
    const unsubscribe = onSnapshot(appsCollectionRef,
(snapshot) => {
      const fetchedApps = snapshot.docs.map(docData => ({
        id: docData.id,
```

```
        ...docData.data()
      }));
      setAvailableApps(fetchedApps);
      setIsLoadingTasks(false); // Done loading
    }, (error) => {
      console.error("Error fetching tasks in UserTaskList:", error);
      setIsLoadingTasks(false);
    });

    return () => unsubscribe(); // Cleanup listener
  }, [db, appId]);

  // handleOpenUploadModal: Sets which app the user wants to upload a screenshot for.
  const handleOpenUploadModal = (appToSelect) => {
    // Check if the user has already completed this specific task.
    const hasAlreadyCompleted = userProfile?.completedTasks?.some(
      (task) => task.appId === appToSelect.id && task.status === 'completed'
    );
```

```
    if (hasAlreadyCompleted) {
      setMessage('You have already successfully completed this task. Great job!');
      setIsMessageModalVisible(true);
      return;
    }
    setSelectedAppForUpload(appToSelect); // Set the app to open the upload modal for
  };

  // handleCloseUploadModal: Closes the screenshot upload modal.
  const handleCloseUploadModal = () => {
    setSelectedAppForUpload(null);
  };

  // handleTaskCompletion: Called when a screenshot is successfully uploaded.
  // It updates the user's profile with earned points and the completed task.
  const handleTaskCompletion = async (completedAppId, completedAppName, earnedPoints, screenshotDownloadUrl) => {
```

```
    if (!userProfile || !currentUserId) {
      setMessage("Error: User profile or ID not available. Please try logging in again.");
      setIsMessageModalVisible(true);
      return;
    }

    // Reference to the user's private profile document.
    // Path: artifacts/{appId}/users/{userId}/profile/data
    const userProfileDocRef = doc(db, `artifacts/${appId}/users/${currentUserId}/profile`, 'data');

    try {
      // Update the user's document in Firestore.
      await updateDoc(userProfileDocRef, {
        points: userProfile.points + earnedPoints, // Add new points to total
        completedTasks: arrayUnion({ // Add this task to the completed tasks array
          appId: completedAppId,
          appName: completedAppName,
          pointsEarned: earnedPoints,
          screenshotUrl: screenshotDownloadUrl,
```

```
        status: 'completed', // Mark the task as completed
        completedAt: new Date() // Record the completion date
      })
    });

    // Also update the local userProfile state in AuthContext to reflect changes immediately.
    updateUserProfile(prevProfile => ({
      ...prevProfile,
      points: prevProfile.points + earnedPoints,
      completedTasks: [...(prevProfile.completedTasks || []), { // Append to the existing array
        appId: completedAppId,
        appName: completedAppName,
        pointsEarned: earnedPoints,
        screenshotUrl: screenshotDownloadUrl,
        status: 'completed',
        completedAt: new Date()
      }]
    }));

    setMessage(`Congratulations! You earned ${earnedPoints} points for "${completedAppName}".`);
```

```
      setIsMessageModalVisible(true);
      handleCloseUploadModal(); // Close the screenshot upload modal
    } catch (error) {
      console.error("Error completing task and updating profile:", error);
      setMessage("Failed to record task completion. Please try again.");
      setIsMessageModalVisible(true);
    }
  };

  if (isLoadingTasks) {
    return <LoadingSpinner />; // Show spinner while tasks are loading
  }

  return (
    <div className="container mx-auto p-6 bg-white rounded-xl shadow-lg mt-8">
      <h2 className="text-3xl font-extrabold text-purple-700 mb-6 text-center">Available Tasks (Apps to Download)</h2>
      {availableApps.length === 0 ? (
```

```
      <p className="text-center text-gray-600 text-lg">No new tasks available right now. Please check back later!</p>
    ) : (
      <div className="grid grid-cols-1 md:grid-cols-2 lg:grid-cols-3 gap-6">
        {availableApps.map((app) => (
          <div key={app.id} className="bg-gray-50 p-6 rounded-lg shadow-md border border-gray-200 flex flex-col justify-between">
            <div>
              <h3 className="text-xl font-bold text-gray-800 mb-2">{app.appName}</h3>
              <p className="text-purple-600 font-semibold mb-3">{app.points} Points</p>
              <p className="text-gray-600 text-sm mb-4">{app.description}</p>
              <a
                href={app.appUrl}
                target="_blank"
                rel="noopener noreferrer"
                className="inline-flex items-center text-blue-600 hover:text-blue-800 font-medium transition duration-200 mb-4"
              >
```

```
                Download App
                <svg xmlns="http://www.w3.org/2000/svg"
width="16" height="16" viewBox="0 0 24 24" fill="none"
stroke="currentColor" strokeWidth="2"
strokeLinecap="round" strokeLinejoin="round"
className="ml-1"><path d="M18 13v6a2 2 0 0 1-2 2H5a2 2
0 0 1-2-2V8a2 2 0 0 1 2-2h6"/><polyline points="15 3 21 3 21
9"/><line x1="10" y1="14" x2="21" y2="3"/></svg>
              </a>
            </div>
            <button
              onClick={() => handleOpenUploadModal(app)}
              // Disable button if user has already completed this
task
              className={`w-full py-2 px-4 rounded-lg font-
semibold transition duration-300 transform hover:scale-105
                ${userProfile?.completedTasks?.some(t => t.appId
=== app.id && t.status === 'completed')
                  ? 'bg-gray-400 text-gray-700 cursor-not-allowed' //
Greyed out if completed
                  : 'bg-green-600 text-white hover:bg-green-700
shadow-md' // Green if not completed
                }`}
              disabled={userProfile?.completedTasks?.some(t =>
t.appId === app.id && t.status === 'completed')}
```

```
            >
              {userProfile?.completedTasks?.some(t => t.appId
=== app.id && t.status === 'completed')
                ? 'Task Completed!'
                : 'Submit Screenshot'
              }
            </button>
          </div>
        ))}
      </div>
    )}
    <div className="mt-8 text-center">
      <button
        onClick={() => setCurrentPage('user-panel')}
        className="text-purple-600 hover:text-purple-800
font-medium hover:underline"
      >
        Back to User Dashboard
      </button>
    </div>
    {/* Conditionally render the ScreenshotUploadModal if an
app is selected */}
    {selectedAppForUpload && (
```

```
        <ScreenshotUploadModal
          app={selectedAppForUpload}
          onClose={handleCloseUploadModal}
          onUploadSuccess={handleTaskCompletion}
        />
      )}
      <MessageModal message={message}
onClose={handleMessageModalClose} />
    </div>
  );
};

// ScreenshotUploadModal: A modal for users to drag & drop
or select a screenshot.
const ScreenshotUploadModal = ({ app, onClose,
onUploadSuccess }) => {
  const [selectedFile, setSelectedFile] = useState(null); // The
file chosen by the user
  const [filePreviewUrl, setFilePreviewUrl] = useState(null); //
URL for displaying the image preview
  const [isDraggingOver, setIsDraggingOver] = useState(false);
// State for drag-and-drop styling
  const [isUploading, setIsUploading] = useState(false); //
State to show 'Uploading...' text
```

```
  // State for messages and modal
  const [message, setMessage] = useState('');
  const [isMessageModalVisible, setIsMessageModalVisible] =
useState(false);

  // Get Firebase Storage and user ID from AuthContext
  const { storage, currentUserId, appId } =
useContext(AuthContext);

  // handleDragOver: Prevents default behavior and sets drag
state for styling.
  const handleDragOver = (event) => {
    event.preventDefault();
    setIsDraggingOver(true);
  };

  // handleDragLeave: Resets drag state when cursor leaves
the drop zone.
  const handleDragLeave = () => {
    setIsDraggingOver(false);
  };

  // handleDrop: Processes the file dropped by the user.
```

```
  const handleDrop = (event) => {
    event.preventDefault();
    setIsDraggingOver(false);
    const droppedFile = event.dataTransfer.files[0]; // Get the first dropped file

    if (droppedFile && droppedFile.type.startsWith('image/')) {
      // If it's an image, set the file and create a preview URL
      setSelectedFile(droppedFile);
      setFilePreviewUrl(URL.createObjectURL(droppedFile));
    } else {
      setMessage('Only image files (like JPG, PNG) are allowed for screenshots.');
      setIsMessageModalVisible(true);
    }
  };

  // handleFileChange: Processes the file selected via the input button.
  const handleFileChange = (event) => {
    const chosenFile = event.target.files[0]; // Get the first selected file
```

```
    if (chosenFile && chosenFile.type.startsWith('image/')) {
      // If it's an image, set the file and create a preview URL
      setSelectedFile(chosenFile);
      setFilePreviewUrl(URL.createObjectURL(chosenFile));
    } else {
      setMessage('Only image files (like JPG, PG) can be uploaded.');
      setIsMessageModalVisible(true);
    }
  };

  // handleUpload: Uploads the selected file to Firebase Storage.
  const handleUpload = async () => {
    if (!selectedFile) {
      setMessage('Please select a screenshot file before uploading.');
      setIsMessageModalVisible(true);
      return;
    }

    setIsUploading(true); // Start showing 'Uploading...'
    setMessage(''); // Clear previous messages
```

```
    try {
        // Create a reference to where the file will be stored in Firebase Storage.
        // Path: artifacts/{appId}/users/{userId}/screenshots/{filename}
        const storageFileRef = ref(storage, `artifacts/${appId}/users/${currentUserId}/screenshots/${selectedFile.name}`);

        // Upload the file bytes.
        const uploadSnapshot = await uploadBytes(storageFileRef, selectedFile);
        // Get the publicly accessible URL for the uploaded file.
        const downloadUrl = await getDownloadURL(uploadSnapshot.ref);

        // Call the success callback function provided by the parent (UserTaskList).
        // This will update the user's points and completed tasks in Firestore.
        onUploadSuccess(app.id, app.appName, app.points, downloadUrl);
```

```
      setMessage('Screenshot uploaded successfully! Points will be added to your profile.');
      setIsMessageModalVisible(true);
      // Reset modal state
      setSelectedFile(null);
      setFilePreviewUrl(null);

    } catch (error) {
      console.error("Error uploading screenshot:", error);
      setMessage('Failed to upload screenshot. Please try again.');
      setIsMessageModalVisible(true);
    } finally {
      setIsUploading(false); // Stop showing 'Uploading...'
    }
  };

  // handleMessageModalClose: Function to close the message modal within this component.
  const handleMessageModalClose = () => {
    setIsMessageModalVisible(false);
    setMessage('');
  };
```

```
  return (
    <div className="fixed inset-0 bg-black bg-opacity-50 flex items-center justify-center z-50 p-4">
      <div className="bg-white p-8 rounded-xl shadow-2xl w-full max-w-lg">
        <h2 className="text-2xl font-bold text-purple-700 mb-6 text-center">Upload Screenshot for "{app.appName}"</h2>
        <div
          className={`border-2 border-dashed rounded-lg p-6 text-center transition-all duration-300
            ${isDraggingOver ? 'border-purple-600 bg-purple-50' : 'border-gray-300 bg-gray-50'}`}
          onDragOver={handleDragOver}
          onDragLeave={handleDragLeave}
          onDrop={handleDrop}
        >
          {filePreviewUrl ? (
            // Show image preview if a file is selected
            <img src={filePreviewUrl} alt="Screenshot Preview" className="max-w-full h-auto mx-auto rounded-md shadow-md mb-4" />
          ) : (
            // Show drag & drop instructions if no file is selected
```

```
          <div className="text-gray-500 mb-4">
            {/* Cloud upload SVG icon */}
            <svg xmlns="http://www.w3.org/2000/svg"
width="48" height="48" viewBox="0 0 24 24" fill="none"
stroke="currentColor" strokeWidth="1.5"
strokeLinecap="round" strokeLinejoin="round"
className="mx-auto mb-2 text-gray-400">
              <path d="M4 14.899A7 7 0 1 1 15.71 8h1.79a4.5 4.5
0 0 1 2.5 8.242"/><path d="M12 12v6"/><path d="m15 15-3
3-3-3"/>
            </svg>
            <p className="text-lg font-medium">Drag & Drop
your screenshot here</p>
            <p className="text-sm">or</p>
          </div>
        )}
        {/* Hidden file input, styled by the label */}
        <input
          type="file"
          accept="image/*" // Only allow image files
          onChange={handleFileChange}
          className="hidden"
          id="file-upload-input"
        />
```

```
        <label
          htmlFor="file-upload-input"
          className="inline-block bg-purple-500 text-white py-2 px-4 rounded-md cursor-pointer hover:bg-purple-600 transition duration-300"
        >
          {filePreviewUrl ? 'Change Screenshot File' : 'Browse Files to Select'}
        </label>
      </div>
      <div className="flex justify-end space-x-4 mt-6">
        <button
          onClick={onClose} // Close the modal (provided by parent)
          className="bg-gray-300 text-gray-800 py-2 px-4 rounded-lg hover:bg-gray-400 transition duration-300"
        >
          Cancel
        </button>
        <button
          onClick={handleUpload}
          disabled={!selectedFile || isUploading} // Disable if no file or already uploading
```

```
          className="bg-green-600 text-white py-2 px-4
rounded-lg shadow-md hover:bg-green-700 transition
duration-300 disabled:opacity-50 disabled:cursor-not-
allowed"
        >
          {isUploading ? 'Uploading...' : 'Upload & Confirm Task'}
        </button>
      </div>
    </div>
    <MessageModal message={message}
onClose={handleMessageModalClose} />
  </div>
 );
};

// --- MAIN APPLICATION COMPONENT ---
// This is the top-level component that stitches all other parts
together.
const App = () => {
 // Get authentication status and user role from AuthContext
 const { currentUser, userRole, loadingAuth } =
useContext(AuthContext);
 // Get current page and function to change page from
PageContext
```

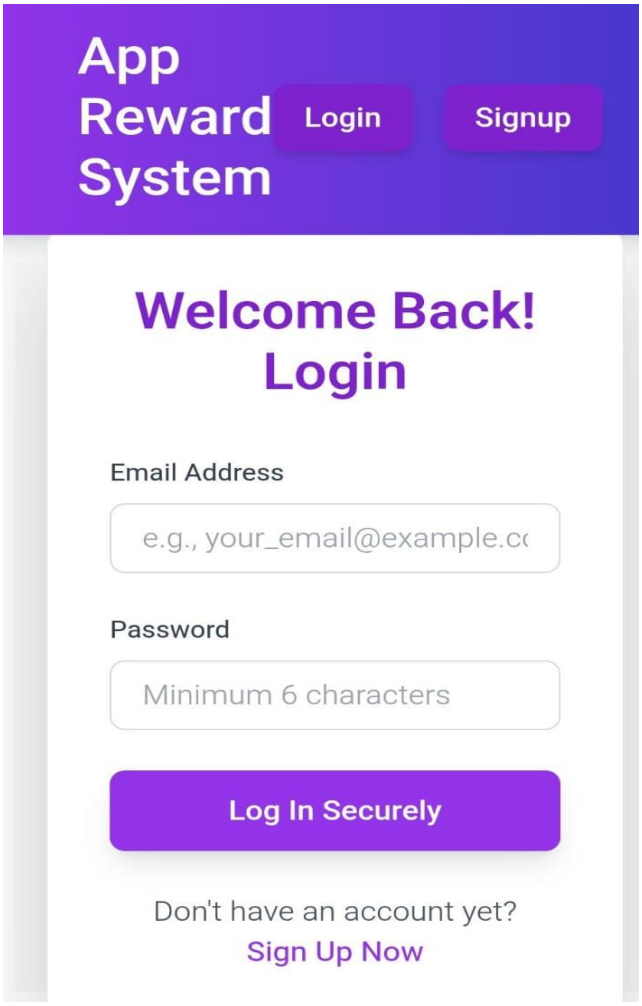
```
  const { currentPage, setCurrentPage } =
useContext(PageContext);


  if (loadingAuth) {
    return <LoadingSpinner />; // Show loading spinner while
authentication is being checked
  }


  // Logic to decide which main page component to render
based on login status and user role.
  let PageToRender;
  if (!currentUser) {
    // If no user is logged in, show Login or Signup form.
    PageToRender = currentPage === 'signup' ? <AuthForm
type="signup" /> : <AuthForm type="login" />;
  } else if (userRole === 'admin') {
    // If an admin is logged in, show admin-specific pages.
    switch (currentPage) {
      case 'admin-add-app':
        PageToRender = <AdminAddApp />;
        break;
      case 'admin-view-apps':
        PageToRender = <AdminViewApps />;
```

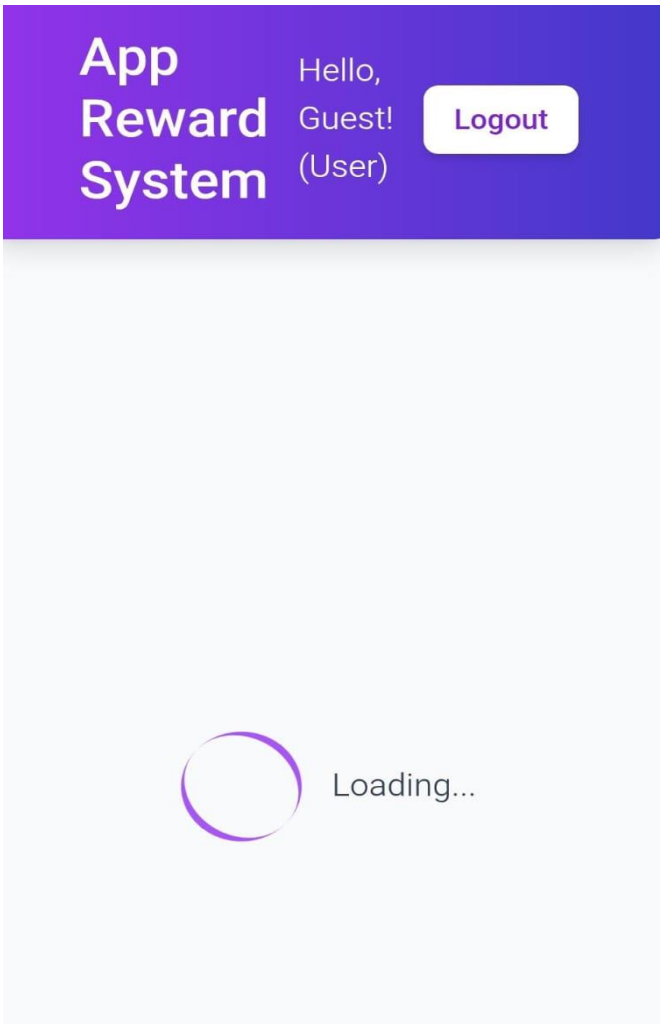
```
        break;
      default:
        // Default admin page is the Admin Panel dashboard.
        PageToRender = <AdminPanel />;
    }
  } else { // userRole === 'user' (or any other role defaults to user)
    // If a regular user is logged in, show user-specific pages.
    switch (currentPage) {
      case 'user-tasks':
        PageToRender = <UserTaskList />;
        break;
      default:
        // Default user page is the User Panel dashboard.
        PageToRender = <UserPanel />;
    }
  }


  return (
    // Apply h-screen and flex-col to make the app fill the whole viewport height
    <div className="h-screen flex flex-col bg-gray-50 font-inter">
```

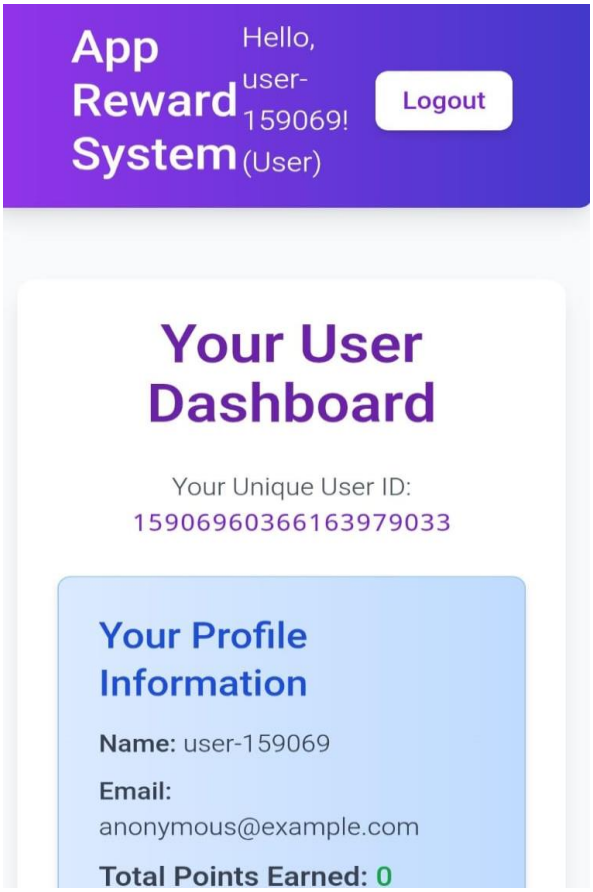
```
      <Header /> {/* The consistent header at the top */}
      {/* The main content area now uses flex-grow to fill remaining space, and has its own padding */}
      <main className="flex-grow overflow-auto p-4">
        {PageToRender} {/* Renders the selected page component */}
      </main>
    </div>
  );
};

// --- ROOT COMPONENT ---
// This is the very first component that gets rendered.
// It sets up the AuthProvider and PageProvider so all other components have access to them.
const Root = () => (
  <AuthProvider>
    <PageProvider>
      <App /> {/* The main application component */}
    </PageProvider>
  </AuthProvider>
);
```

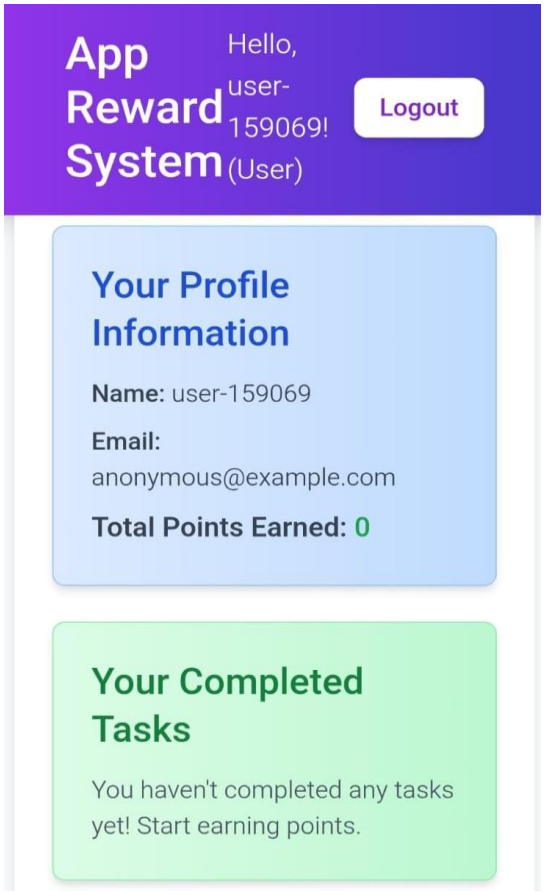
export default Root;

// Export the Root component as the default for the application.









Problem Set 3

A. Write and share a small note about your choice of system to schedule periodic tasks (such as downloading a list of ISINs every 24 hours). Why did you choose it? Is it reliable enough; Or will it scale? If not, what are the problems with it? And, what else would you recommend to fix this problem at scale in production?

Solution:-

The current React application you see in the Canvas is a **front-end application**. This means it runs in the user's web browser and is primarily responsible for the user interface, handling user interactions, and communicating with the Firebase backend (Firestore and Storage). It **does not** have the capability to schedule or execute tasks automatically on a server, such as downloading data every 24 hours. Browser-based JavaScript cannot run continuously in the background when the browser is closed, nor can it reliably access external resources like financial data feeds directly in a production environment due to security and origin policies.

Easy with Firebase: Since app already uses Firebase (for login, user data, app lists, screenshots), Functions are built to work perfectly with it. Your function can easily talk to your Firestore database or Firebase Storage.

B. In what circumstances would you use Flask instead of Django and vice versa?

Solution :-

When choosing between Flask and Django, both excellent Python web frameworks, the decision often comes down to the project's scale, complexity, and the level of control and flexibility you desire. They cater to different needs and development philosophies.

Circumstances for choosing Flask:

1. Small to Medium-Sized Projects.
2. Maximum Flexibility and Control
3. Learning and Teaching
4. Performance Optimization

Circumstances for choosing Django

1. Large and Complex Web Applications
2. Convention Over Configuration
3. Built-in Features and Tools