■ Project Code Export

■ Frontend File List:

- .gitignore
- Frontend_Code_Export.pdf
- db.json
- postcss.config.js
- public\favicon.ico
- public\index.html
- public\left.png
- public\logo192.png
- public\logo512.png
- public\manifest.json
- public\right.png
- public\robots.txt
- script.py
- src\App.css
- src\App.js
- src\App.test.js
- src\AppContent.js
- src\assets\placeholder.svg
- src\components\admin\AdminPanel.js
- src\components\common\LoadingSpinner.js
- src\components\common\ResetAll.js
- src\components\common\TabNavigation.js
- src\components\common\Toasts.js
- src\components\form\FieldInputRow.js
- src\components\form\InputForm.js
- src\components\report\PDFPreview.js
- src\components\report\ReportOutput.js
- src\components\settings\CategoryManager.js
- src\components\settings\SettingsPanel.js
- src\contexts\FormConfigContext.js
- src\contexts\ThemeContext.js
- src\hooks\useConfigImportExport.js
- src\hooks\useExcelExport.js
- src\hooks\usePDFGeneration.js
- src\hooks\useReportGeneration.js
- src\index.css
- src\index.js
- src\logo.svg
- src\reportWebVitals.js
- src\setupTests.js
- src\utils\constants.js
- src\utils\helpers.js
- tailwind.config.js

■ File: .gitignore

```
================================================================================
[Binary file -  format]

--------------------------------------------------------------------------------
```

■ File: Frontend_Code_Export.pdf

```
================================================================================
[Binary file - .pdf format]

--------------------------------------------------------------------------------
```

■ File: db.json

```
================================================================================
```


```
  1: {
  2:   "settings": {
  3:     "title": "GENOMICS & DIET",
  4:     "quote": "\"YOU ARE WHAT YOU EAT\" - Victor Lindlahr",
  5:     "description": "A good diet is the one that makes you feel happy, keeps you healthy, does not make you
  6:     "headerColor": "#16a34a",
  7:     "highThreshold": 6,
  8:     "colors": {
  9:       "low": "#16a34a",
 10:       "medium": "#f59e0b",
 11:       "high": "#dc2626"
 12:     }
 13:   },
 14:   "categories": [
 15:     {
 16:       "id": "1",
 17:       "name": "MACRONUTRIENTS"
 18:     },
 19:     {
 20:       "id": "2",
 21:       "name": "MEAL PATTERN"
 22:     },
 23:     {
 24:       "id": "3",
 25:       "name": "FOOD SENSITIVITIES"
 26:     },
 27:     {
 28:       "id": "29b9",
 29:       "name": "hello"
 30:     }
 31:   ],
 32:   "fields": [
 33:     {
 34:       "id": "field_carb_1752645546685",
 35:       "_uuid": "01a65289-f147-459d-bb9d-3fded0529b76",
 36:       "_originalId": "field_carb_1752645546685",
 37:       "label": "carb",
 38:       "category": "MACRONUTRIENTS",
 39:       "min": 1,
 40:       "max": 10,
 41:       "high": "Maintain carb intake 40% For obesity & IR control",
 42:       "normal": "Maintain carb intake 50%Balanced recommendation",
 43:       "low": "Maintain carb intake 60%For obesity & IR control"
 44:     },
 45:     {
 46:       "id": "field_fat_sensitivity_1752645768322",
 47:       "_uuid": "1124cfb9-2ac0-407f-8677-14d08aaa9143",
 48:       "_originalId": "field_fat_sensitivity_1752645768322",
 49:       "label": "Fat Sensitivity",
 50:       "category": "MACRONUTRIENTS",
 51:       "min": 1,
 52:       "max": 10,
 53:       "high": "Fat intake not to exceed\\n15% of total calories",
 54:       "normal": "Fat intake should be\\n20% of total calories",
 55:       "low": "Fat intake advised up to\\n25% of total calories"
 56:     },
 57:     {
 58:       "id": "field_protein_requirement_1752645831504",
 59:       "_uuid": "f3d9af18-232e-4466-b0e5-7cdd92a8e477",
```

```
      "_originalId": "field_protein_requirement_1752645831504",
      "label": "Protein Requirement",
      "category": "MACRONUTRIENTS",
      "min": 1,
      "max": 10,
      "high": "Protein supplements needed along with dietary source",
      "normal": "Maintain adequate protein through balanced diet and occasional supplements",
      "low": "Protein supplements not needed, intake through diet is enough"
    },
    {
      "id": "field_meal_frequency_1752645937311",
      "_uuid": "7afd567a-4a36-42f9-9f4f-2fdf9a95e0e2",
      "_originalId": "field_meal_frequency_1752645937311",
      "label": "Meal Frequency",
      "category": "MEAL PATTERN",
      "min": 1,
      "max": 10,
      "high": "4-5 small meals suggested in a day",
      "normal": "3-4 balanced meals recommended per day",
      "low": "Less frequent meals, 2-3 meals are enough in a day"
    },
    {
      "id": "field_alcohol_sensitivity_1752646186761",
      "_uuid": "caa203af-d06f-4a99-a7fc-b5665c9c6f82",
      "_originalId": "field_alcohol_sensitivity_1752646186761",
      "label": "Alcohol Sensitivity",
      "category": "",
      "min": 1,
      "max": 10,
      "high": "High sensitivity, avoid alcohol, if possible, especially the types of beverages that trigger
      "normal": "Moderate sensitivity, limit alcohol consumption to 1-2 drinks per day, monitor for any adv
      "low": "Low sensitivity, know your alcohol intake limits, consult doctor for knowing your upper limit
    },
    {
      "id": "field_caffeine_sensitivity_1752646351914",
      "_uuid": "c14ecc31-3c3f-4b89-bdb4-67f9a0df6e8d",
      "_originalId": "field_caffeine_sensitivity_1752646351914",
      "label": "Caffeine Sensitivity",
      "category": "",
      "min": 1,
      "max": 10,
      "high": "High sensitivity, do not consume >4 cups/day",
      "normal": "Moderate sensitivity, limit caffeine to 4-5 cups per day",
      "low": "Caffeine up to 5 cups a day can be consumed"
    },
    {
      "id": "field_gluten_sensitivity_1752646517788",
      "_uuid": "af6eb4cb-15ca-4f5c-9c31-07ce400aa8aa",
      "_originalId": "field_gluten_sensitivity_1752646517788",
      "label": "Gluten Sensitivity",
      "category": "FOOD SENSITIVITIES",
      "min": 1,
      "max": 10,
      "high": "Gluten intake needs to be reduced stopped",
      "normal": "Monitor gluten intake, reduce if experiencing digestive issues",
      "low": "Gluten to be avoided in cases of gastric distress"
    },
    {
      "id": "field_salt_sensitivity_1752646658680",
      "_uuid": "4086fe30-a94e-43b9-914d-7dd032448a7c",
      "_originalId": "field_salt_sensitivity_1752646658680",
      "label": "Salt Sensitivity",
      "category": "",
      "min": 1,
      "max": 10,
      "high": "Try to reduce overall salt intake to up to 3-5 gm per day",
      "normal": "Maintain salt intake around 5 gm per day",
      "low": "Consumption of salt up to 5 gm/day can be done"
    }
  ]
}
```

■ File: postcss.config.js

```
  1: module.exports = {
  2:   plugins: {
  3:     tailwindcss: {},
  4:     autoprefixer: {},
  5:   },
  6: }
```

■ File: public\favicon.ico

[Binary file - .ico format]

■ File: public\index.html

```
  1: <!DOCTYPE html>
  2: <html lang="en">
  3:   <head>
  4:     <meta charset="utf-8" />
  5:     <link rel="icon" href="%PUBLIC_URL%/favicon.ico" />
  6:     <meta name="viewport" content="width=device-width, initial-scale=1" />
  7:     <meta name="theme-color" content="#000000" />
  8:     <meta
  9:       name="description"
 10:       content="Web site created using create-react-app"
 11:     />
 12:     <link rel="apple-touch-icon" href="%PUBLIC_URL%/logo192.png" />
 13:     <!--
 14:       manifest.json provides metadata used when your web app is installed on a
 15:       user's mobile device or desktop. See https://developers.google.com/web/fundamentals/web-app-manifest/
 16:     -->
 17:     <link rel="manifest" href="%PUBLIC_URL%/manifest.json" />
 18:     <!--
 19:       Notice the use of %PUBLIC_URL% in the tags above.
 20:       It will be replaced with the URL of the `public` folder during the build.
 21:       Only files inside the `public` folder can be referenced from the HTML.
 22:
 23:       Unlike "/favicon.ico" or "favicon.ico", "%PUBLIC_URL%/favicon.ico" will
 24:       work correctly both with client-side routing and a non-root public URL.
 25:       Learn how to configure a non-root public URL by running `npm run build`.
 26:     -->
 27:     <title>React App</title>
 28:   </head>
 29:   <body>
 30:     <noscript>You need to enable JavaScript to run this app.</noscript>
 31:     <div id="root"></div>
 32:     <!--
 33:       This HTML file is a template.
 34:       If you open it directly in the browser, you will see an empty page.
 35:
 36:       You can add webfonts, meta tags, or analytics to this file.
 37:       The build step will place the bundled scripts into the <body> tag.
 38:
 39:       To begin the development, run `npm start` or `yarn start`.
 40:       To create a production bundle, use `npm run build` or `yarn build`.
 41:     -->
 42:   </body>
 43: </html>
```

■ File: public\left.png

[Binary file - .png format]

■ File: public\logo192.png

```
================================================================================
[Binary file - .png format]

--------------------------------------------------------------------------------
```

■ File: public\logo512.png

```
================================================================================
[Binary file - .png format]

--------------------------------------------------------------------------------
```

■ File: public\manifest.json

```
================================================================================
  1: {
  2:   "short_name": "React App",
  3:   "name": "Create React App Sample",
  4:   "icons": [
  5:     {
  6:       "src": "favicon.ico",
  7:       "sizes": "64x64 32x32 24x24 16x16",
  8:       "type": "image/x-icon"
  9:     },
 10:     {
 11:       "src": "logo192.png",
 12:       "type": "image/png",
 13:       "sizes": "192x192"
 14:     },
 15:     {
 16:       "src": "logo512.png",
 17:       "type": "image/png",
 18:       "sizes": "512x512"
 19:     }
 20:   ],
 21:   "start_url": ".",
 22:   "display": "standalone",
 23:   "theme_color": "#000000",
 24:   "background_color": "#ffffff"
 25: }

--------------------------------------------------------------------------------
```

■ File: public\right.png

```
================================================================================
[Binary file - .png format]

--------------------------------------------------------------------------------
```

■ File: public\robots.txt

```
================================================================================
# https://www.robotstxt.org/robotstxt.html
User-agent: *
Disallow:

--------------------------------------------------------------------------------
```

■ File: script.py

```
================================================================================
  1: import os
  2: from reportlab.lib.pagesizes import A4
  3: from reportlab.lib.units import mm
  4: from reportlab.pdfgen import canvas
  5:
  6: def get_code_files(directory, excluded_files=None, excluded_dirs=None):
  7:     """Fetch all project files except specified exclusions."""
  8:     if excluded_files is None:
  9:         excluded_files = {'package.json', 'package-lock.json'}
 10:
 11:     if excluded_dirs is None:
 12:         excluded_dirs = {'node_modules', '.git', '__pycache__', 'build', '.next', 'dist'}
 13:
 14:     code_files = {}
 15:
```

```
    for root, dirs, files in os.walk(directory):
        # Skip excluded directories
        dirs[:] = [d for d in dirs if d not in excluded_dirs]

        # Skip if current directory is an excluded directory
        if any(excluded_dir in root.split(os.sep) for excluded_dir in excluded_dirs):
            continue

        for file in files:
            # Skip excluded files
            if file in excluded_files:
                continue

            file_path = os.path.join(root, file)

            # Get file extension
            _, ext = os.path.splitext(file)

            try:
                # Try to read as text file first
                if ext.lower() in {'.js', '.jsx', '.ts', '.tsx', '.css', '.scss', '.sass', '.less',
                                   '.html', '.htm', '.json', '.md', '.txt', '.xml', '.yaml', '.yml',
                                   '.config', '.gitignore', '.env', '.py', '.sh', '.bat', '.cmd',
                                   '.svg', '.dockerfile', '.editorconfig', '.eslintrc', '.prettierrc'}:
                    with open(file_path, "r", encoding="utf-8", errors="ignore") as f:
                        code_files[file_path] = f.readlines()
                else:
                    # For binary files, just note them as binary
                    code_files[file_path] = [f"[Binary file - {ext} format]"]

            except Exception as e:
                print(f"? Error reading {file_path}: {e}")
                code_files[file_path] = [f"[Error reading file: {str(e)}]"]

    return code_files


def create_pdf(code_data, output_pdf="Frontend_Code_Export.pdf"):
    c = canvas.Canvas(output_pdf, pagesize=A4)
    width, height = A4
    margin = 20 * mm
    line_height = 10
    y = height - margin

    # Title
    c.setFont("Helvetica-Bold", 16)
    c.drawString(margin, y, "? Project Code Export")
    y -= 2 * line_height
    c.setFont("Helvetica-Bold", 12)
    c.drawString(margin, y, "? Frontend File List:")
    y -= 2 * line_height

    file_paths = sorted(list(code_data.keys()))

    # 1. File list (original simple format)
    c.setFont("Courier", 8)
    for path in file_paths:
        if y < margin:
            c.showPage()
            c.setFont("Courier", 8)
            y = height - margin

        display_path = os.path.relpath(path)
        c.drawString(margin, y, f"- {display_path}")
        y -= line_height

    # Add page break before code content
    c.showPage()
    y = height - margin

    # 2. File contents
    for file_path in file_paths:
        lines = code_data[file_path]
```

```
        print(f"? Adding: {file_path}")

        if y < margin + 3 * line_height:
            c.showPage()
            y = height - margin

        # File header
        rel_path = os.path.relpath(file_path)
        c.setFont("Helvetica-Bold", 12)
        c.drawString(margin, y, f"? File: {rel_path}")
        y -= line_height

        # Add separator line
        c.setFont("Courier", 8)
        c.drawString(margin, y, "=" * 80)
        y -= line_height

        # File content
        for line_num, line in enumerate(lines, 1):
            if y < margin:
                c.showPage()
                c.setFont("Courier", 8)
                y = height - margin

            # Clean and truncate line
            line = line.strip("\n").encode("latin-1", "replace").decode("latin-1")

            # Add line numbers for code files
            if rel_path.endswith(('.js', '.jsx', '.ts', '.tsx', '.css', '.py', '.html', '.json')):
                display_line = f"{line_num:3d}: {line[:280]}"
            else:
                display_line = line[:300]

            c.drawString(margin, y, display_line)
            y -= line_height

        # Add spacing between files
        y -= line_height
        if y > margin:
            c.setFont("Courier", 8)
            c.drawString(margin, y, "-" * 80)
            y -= 2 * line_height

    c.save()
    print(f"? PDF successfully created: {output_pdf}")
    print(f"? Total files processed: {len(code_data)}")


def main():
    root_dir = os.path.dirname(os.path.abspath(__file__))

    # Files to exclude (including package.json as requested)
    excluded_files = {
        'package.json',
        'package-lock.json',
        'yarn.lock',
        'README.md',
        '.DS_Store',
        'Thumbs.db',
        'Desktop.ini'
    }

    # Directories to exclude
    excluded_dirs = {
        'node_modules',
        '.git',
        '__pycache__',
        'build',
        'dist',
        '.next',
        'coverage',
        '.nyc_output',
        'logs',
```

```
162:         '*.log'
163:     }
164: 
165:     print("? Scanning project files...")
166:     code_files = get_code_files(root_dir, excluded_files, excluded_dirs)
167: 
168:     if not code_files:
169:         print("? No files found to process!")
170:         return
171: 
172:     print(f"? Found {len(code_files)} files to include in PDF")
173:     create_pdf(code_files)
174: 
175: 
176: if __name__ == "__main__":
177:     main()
```

■ File: src\App.css

```
  1: .App {
  2:   text-align: center;
  3: }
  4: 
  5: .App-logo {
  6:   height: 40vmin;
  7:   pointer-events: none;
  8: }
  9: 
 10: @media (prefers-reduced-motion: no-preference) {
 11:   .App-logo {
 12:     animation: App-logo-spin infinite 20s linear;
 13:   }
 14: }
 15: 
 16: .App-header {
 17:   background-color: #282c34;
 18:   min-height: 100vh;
 19:   display: flex;
 20:   flex-direction: column;
 21:   align-items: center;
 22:   justify-content: center;
 23:   font-size: calc(10px + 2vmin);
 24:   color: white;
 25: }
 26: 
 27: .App-link {
 28:   color: #61dafb;
 29: }
 30: 
 31: @keyframes App-logo-spin {
 32:   from {
 33:     transform: rotate(0deg);
 34:   }
 35:   to {
 36:     transform: rotate(360deg);
 37:   }
 38: }
```

■ File: src\App.js

```
  1: import React from 'react';
  2: import { FormConfigProvider } from './contexts/FormConfigContext';
  3: import { ThemeProvider } from './contexts/ThemeContext';
  4: import AppContent from './AppContent';
  5: import Toasts from './components/common/Toasts';
  6: // import { FormConfigProvider } from "./contexts/FormConfigContext";
  7: 
  8: 
  9: function App() {
```

```
 10:   return (
 11:     <ThemeProvider>
 12:       <FormConfigProvider>
 13:         <AppContent />
 14:         <Toasts />
 15:       </FormConfigProvider>
 16:     </ThemeProvider>
 17:   );
 18: }
 19:
 20: export default App;
```

■ File: src\App.test.js

```
  1: import { render, screen } from '@testing-library/react';
  2: import App from './App';
  3:
  4: test('renders learn react link', () => {
  5:   render(<App />);
  6:   const linkElement = screen.getByText(/learn react/i);
  7:   expect(linkElement).toBeInTheDocument();
  8: });
```

■ File: src\AppContent.js

```
  1: import React, { useState, useEffect } from "react";
  2: import TabNavigation from "./components/common/TabNavigation";
  3: import InputForm from "./components/form/InputForm";
  4: import ReportOutput from "./components/report/ReportOutput";
  5: import AdminPanel from "./components/admin/AdminPanel";
  6: import SettingsPanel from "./components/settings/SettingsPanel";
  7: import { useReportGeneration } from "./hooks/useReportGeneration";
  8: import { AnimatePresence, motion } from "framer-motion";
  9:
 10: const LOCAL_TAB_KEY = "activeTab";
 11:
 12: const AppContent = () => {
 13:   // ? Load initial tab from localStorage if available
 14:   const [activeTab, setActiveTab] = useState(() => {
 15:     return localStorage.getItem(LOCAL_TAB_KEY) || "form";
 16:   });
 17:
 18:   const { reportData, generateReport } = useReportGeneration();
 19:
 20:   // ? Sync tab state to localStorage on every change
 21:   useEffect(() => {
 22:     localStorage.setItem(LOCAL_TAB_KEY, activeTab);
 23:   }, [activeTab]);
 24:
 25:   const handleGenerateReport = (formData) => {
 26:     generateReport(formData);
 27:   };
 28:
 29:   const tabTransition = {
 30:     initial: { opacity: 0, y: 20 },
 31:     animate: { opacity: 1, y: 0 },
 32:     exit: { opacity: 0, y: -20 },
 33:     transition: { duration: 0.3 },
 34:   };
 35:
 36:   return (
 37:     <div className="bg-gray-100 font-sans min-h-screen dark:bg-gray-900">
 38:       <TabNavigation activeTab={activeTab} setActiveTab={setActiveTab} />
 39:
 40:       <AnimatePresence mode="wait">
 41:         {activeTab === "form" && (
 42:           <motion.div key="form" {...tabTransition}>
 43:             <div className="flex flex-col md:flex-row h-screen desktop-layout">
 44:               <InputForm
```

```
45:                 onGenerateReport={handleGenerateReport}
46:                 reportData={reportData}
47:               />
48:               <ReportOutput reportData={reportData} />
49:             </div>
50:           </motion.div>
51:         )}
52:
53:         {activeTab === "admin" && (
54:           <motion.div key="admin" {...tabTransition}>
55:             <div className="p-6">
56:               <AdminPanel />
57:             </div>
58:           </motion.div>
59:         )}
60:
61:         {activeTab === "settings" && (
62:           <motion.div key="settings" {...tabTransition}>
63:             <div className="p-6">
64:               <SettingsPanel />
65:             </div>
66:           </motion.div>
67:         )}
68:       </AnimatePresence>
69:     </div>
70:   );
71: };
72:
73: export default AppContent;
```

■ File: src\assets\placeholder.svg

==========

■ File: src\components\admin\AdminPanel.js

```
  1: import React, { useEffect, useState } from "react";
  2: import { DragDropContext, Droppable, Draggable } from "@hello-pangea/dnd";
  3: import { v4 as uuidv4 } from "uuid";
  4:
  5: const API_URL = "http://localhost:5000";
  6:
  7: const AdminPanel = () => {
  8:   const [localFields, setLocalFields] = useState([]);
  9:   const [categories, setCategories] = useState([]);
 10:   const [loading, setLoading] = useState(true);
 11:   const [error, setError] = useState(null);
 12:   const [showCreateModal, setShowCreateModal] = useState(false);
 13:   const [newFieldData, setNewFieldData] = useState({
 14:     label: "",
 15:     category: "",
 16:     min: 1,
 17:     max: 10,
 18:   });
 19:
 20:   // Toast simulation (since we can't use react-toastify)
 21:   const showToast = (message, type = "info") => {
 22:     console.log(`${type.toUpperCase()}: ${message}`);
 23:     // In a real app, you'd use a proper toast library
 24:     alert(message);
 25:   };
 26:
 27:   // Load fields
 28:   const fetchFields = async () => {
 29:     try {
 30:       setLoading(true);
 31:       setError(null);
 32:       const res = await fetch(`${API_URL}/fields`);
 33:
 34:       if (!res.ok) {
```

```
        throw new Error(`HTTP error! status: ${res.status}`);
      }

      let data = await res.json();
      data = data.map((f) => ({
        ...f,
        _uuid: uuidv4(),
        _originalId: f.id,
      }));
      setLocalFields(data);
    } catch (err) {
      const errorMsg = "? Failed to load fields";
      setError(errorMsg);
      showToast(errorMsg, "error");
      console.error(err);
    } finally {
      setLoading(false);
    }
  };

  // Load categories
  const fetchCategories = async () => {
    try {
      const res = await fetch(`${API_URL}/categories`);

      if (!res.ok) {
        throw new Error(`HTTP error! status: ${res.status}`);
      }

      const data = await res.json();
      const names = data.map((cat) =>
        typeof cat === "string" ? cat : cat.name
      );
      setCategories(names);
    } catch (err) {
      const errorMsg = "? Failed to load categories";
      showToast(errorMsg, "error");
      console.error(err);
    }
  };

  useEffect(() => {
    fetchFields();
    fetchCategories();
  }, []);

  const addNewField = async (e) => {
    if (e) {
      e.preventDefault();
      e.stopPropagation();
    }

    if (!newFieldData.label.trim()) {
      showToast("? Field label is required", "error");
      return;
    }

    // Check for duplicate field IDs based on label
    const fieldId = `field_${newFieldData.label
      .toLowerCase()
      .replace(/\s+/g, "_")}_${Date.now()}`;
    const isDuplicate = localFields.some(
      (field) =>
        field.label.toLowerCase() === newFieldData.label.toLowerCase().trim()
    );

    if (isDuplicate) {
      showToast("? A field with this name already exists", "error");
      return;
    }

    const newField = {
      _uuid: uuidv4(),
```

```
      _originalId: null,
      id: fieldId,
      label: newFieldData.label.trim(),
      category: newFieldData.category,
      min: Number(newFieldData.min),
      max: Number(newFieldData.max),
      high: "",
      normal: "",
      low: "",
    };

    try {
      const res = await fetch(`${API_URL}/fields`, {
        method: "POST",
        headers: { "Content-Type": "application/json" },
        body: JSON.stringify(newField),
      });

      if (res.ok) {
        showToast("? Field added!", "success");
        setShowCreateModal(false);
        setNewFieldData({ label: "", category: "", min: 1, max: 10 });
        fetchFields();
      } else {
        throw new Error(`HTTP error! status: ${res.status}`);
      }
    } catch (err) {
      showToast("? Failed to add field", "error");
      console.error(err);
    }
  };

  const handleCreateModalClose = () => {
    setShowCreateModal(false);
    setNewFieldData({ label: "", category: "", min: 1, max: 10 });
  };

  const saveField = async (index) => {
    const field = localFields[index];
    const targetId = field._originalId ?? field.id;

    try {
      const res = await fetch(`${API_URL}/fields/${targetId}`, {
        method: "PUT",
        headers: { "Content-Type": "application/json" },
        body: JSON.stringify({ ...field }),
      });

      if (res.ok) {
        showToast("? Field saved", "success");
        fetchFields();
      } else {
        throw new Error(`HTTP error! status: ${res.status}`);
      }
    } catch (err) {
      showToast("? Failed to save field", "error");
      console.error(err);
    }
  };

  const deleteField = async (id) => {
    if (!window.confirm("Are you sure?")) return;

    try {
      const res = await fetch(`${API_URL}/fields/${id}`, { method: "DELETE" });

      if (res.ok || res.status === 404) {
        showToast("?? Field deleted", "info");
        fetchFields();
      } else {
        throw new Error(`HTTP error! status: ${res.status}`);
      }
    } catch (err) {
```

```
      showToast("? Failed to delete field", "error");
      console.error(err);
    }
  };

  const updateLocalField = (index, key, value) => {
    const updated = [...localFields];

    if (key === "id") {
      const isDuplicate = localFields.some(
        (f, i) => i !== index && f.id.trim() === value.trim()
      );
      if (isDuplicate) {
        showToast("? Field ID must be unique", "error");
        return;
      }
    }

    // Handle number inputs properly
    if (key === "min" || key === "max") {
      const numValue = parseInt(value);
      if (isNaN(numValue)) {
        value = key === "min" ? 1 : 10;
      } else {
        value = numValue;
      }
    }

    updated[index] = { ...updated[index], [key]: value };
    setLocalFields(updated);
  };

  const onDragEnd = (result) => {
    if (!result.destination) return;

    const reordered = [...localFields];
    const [moved] = reordered.splice(result.source.index, 1);
    reordered.splice(result.destination.index, 0, moved);
    setLocalFields(reordered);
    showToast("? Reordered (not saved)", "info");
  };

  if (loading) {
    return (
      <div className="min-h-screen flex items-center justify-center">
        <div className="text-center">
          <div className="animate-spin rounded-full h-12 w-12 border-b-2 border-blue-600 mx-auto mb-4"></div
          <p className="text-gray-600">Loading fields...</p>
        </div>
      </div>
    );
  }

  if (error) {
    return (
      <div className="min-h-screen flex items-center justify-center">
        <div className="text-center">
          <div className="text-red-500 text-xl mb-4">?? Error</div>
          <p className="text-gray-600 mb-4">{error}</p>
          <button
            onClick={() => {
              fetchFields();
              fetchCategories();
            }}
            className="bg-blue-600 hover:bg-blue-700 text-white px-4 py-2 rounded-lg"
          >
            Try Again
          </button>
        </div>
      </div>
    );
  }

```

```
  return (
    <div className="min-h-screen bg-gray-50">
      <div className="max-w-7xl mx-auto px-4 sm:px-6 lg:px-8 py-4 sm:py-6 lg:py-8">
        <div className="flex flex-col sm:flex-row justify-between items-start sm:items-center mb-6 gap-4">
          <h2 className="text-xl sm:text-2xl font-bold text-gray-900">
            Field Management
          </h2>
          <button
            onClick={() => setShowCreateModal(true)}
            className="w-full sm:w-auto bg-green-600 hover:bg-green-700 text-white px-4 sm:px-5 py-2 rounde
          >
            <span>Add New Field</span>
          </button>
        </div>

        {localFields.length === 0 ? (
          <div className="text-center py-12">
            <p className="text-gray-600 mb-4">No fields available</p>
            <button
              onClick={() => setShowCreateModal(true)}
              className="bg-green-600 hover:bg-green-700 text-white px-6 py-3 rounded-lg font-semibold"
            >
              Create First Field
            </button>
          </div>
        ) : (
          <DragDropContext onDragEnd={onDragEnd}>
            <Droppable droppableId="fields">
              {(provided) => (
                <div ref={provided.innerRef} {...provided.droppableProps}>
                  {localFields.map((field, index) => (
                    <Draggable
                      key={field._uuid}
                      draggableId={field._uuid}
                      index={index}
                    >
                      {(provided, snapshot) => (
                        <div
                          ref={provided.innerRef}
                          {...provided.draggableProps}
                          {...provided.dragHandleProps}
                          className={`border rounded-lg p-4 sm:p-6 bg-white shadow-sm space-y-4 mb-4 sm:mb-
                            snapshot.isDragging
                              ? "bg-gray-100 ring-2 ring-green-500 shadow-lg"
                              : "hover:shadow-md"
                          }`}
                        >
                          <div className="flex flex-col sm:flex-row justify-between items-start sm:items-ce
                            <h3 className="text-lg font-semibold text-gray-900 break-words">
                              {index + 1}. {field.label}
                            </h3>
                            <div className="flex flex-col sm:flex-row gap-2 w-full sm:w-auto">
                              <button
                                onClick={() => saveField(index)}
                                className="w-full sm:w-auto px-3 py-1 bg-blue-600 text-white text-sm rounde
                              >
                                <span>Save</span>
                              </button>
                              <button
                                onClick={() =>
                                  deleteField(field._originalId ?? field.id)
                                }
                                className="w-full sm:w-auto px-3 py-1 bg-red-600 text-white text-sm rounded
                              >
                                <span>Delete</span>
                              </button>
                            </div>
                          </div>

                          <div className="grid grid-cols-1 sm:grid-cols-2 lg:grid-cols-3 gap-4">
                            <FieldInput
                              label="Field ID"
                              value={field.id}
```

```
                              disabled={!!field._originalId}
                              onChange={(val) =>
                                updateLocalField(index, "id", val.trim())
                              }
                            />
                            <FieldInput
                              label="Label"
                              value={field.label}
                              onChange={(val) =>
                                updateLocalField(index, "label", val)
                              }
                            />
                            <FieldSelect
                              label="Category"
                              value={field.category}
                              options={categories}
                              onChange={(val) =>
                                updateLocalField(index, "category", val)
                              }
                            />
                            <FieldInput
                              label="Min"
                              type="number"
                              value={field.min}
                              onChange={(val) =>
                                updateLocalField(index, "min", val)
                              }
                            />
                            <FieldInput
                              label="Max"
                              type="number"
                              value={field.max}
                              onChange={(val) =>
                                updateLocalField(index, "max", val)
                              }
                            />
                            <div className="sm:col-span-2 lg:col-span-3">
                              <div className="grid grid-cols-1 lg:grid-cols-3 gap-4">
                                <FieldTextarea
                                  label="High Recommendation"
                                  value={field.high}
                                  onChange={(val) =>
                                    updateLocalField(index, "high", val)
                                  }
                                />
                                <FieldTextarea
                                  label="Normal Recommendation"
                                  value={field.normal}
                                  onChange={(val) =>
                                    updateLocalField(index, "normal", val)
                                  }
                                />
                                <FieldTextarea
                                  label="Low Recommendation"
                                  value={field.low}
                                  onChange={(val) =>
                                    updateLocalField(index, "low", val)
                                  }
                                />
                              </div>
                            </div>
                          </div>
                        </div>
                      )}
                    </Draggable>
                  ))}
                  {provided.placeholder}
                </div>
              )}
            </Droppable>
          </DragDropContext>
        )}
      </div>
```

```
400:
401:       {/* CREATE MODAL - This was missing! */}
402:       {showCreateModal && (
403:         <div className="fixed inset-0 bg-black bg-opacity-50 flex items-center justify-center z-50 p-4">
404:           <div className="bg-white rounded-lg p-6 w-full max-w-md max-h-full overflow-y-auto">
405:             <div className="flex justify-between items-center mb-4">
406:               <h3 className="text-lg font-semibold text-gray-900">
407:                 Add New Field
408:               </h3>
409:               <button
410:                 onClick={handleCreateModalClose}
411:                 className="text-gray-400 hover:text-gray-600 text-xl"
412:               >
413:                 ?
414:               </button>
415:             </div>
416:
417:             <div className="space-y-4">
418:               <FieldInput
419:                 label="Field Label"
420:                 value={newFieldData.label}
421:                 onChange={(val) =>
422:                   setNewFieldData({ ...newFieldData, label: val })
423:                 }
424:                 placeholder="Enter field label"
425:                 required
426:               />
427:
428:               <FieldSelect
429:                 label="Category"
430:                 value={newFieldData.category}
431:                 options={categories}
432:                 onChange={(val) =>
433:                   setNewFieldData({ ...newFieldData, category: val })
434:                 }
435:               />
436:
437:               <div className="grid grid-cols-2 gap-4">
438:                 <FieldInput
439:                   label="Min Value"
440:                   type="number"
441:                   value={newFieldData.min}
442:                   onChange={(val) =>
443:                     setNewFieldData({ ...newFieldData, min: val })
444:                   }
445:                   min="1"
446:                 />
447:                 <FieldInput
448:                   label="Max Value"
449:                   type="number"
450:                   value={newFieldData.max}
451:                   onChange={(val) =>
452:                     setNewFieldData({ ...newFieldData, max: val })
453:                   }
454:                   min="1"
455:                 />
456:               </div>
457:
458:               <div className="flex flex-col sm:flex-row gap-3 pt-4">
459:                 <button
460:                   type="button"
461:                   onClick={handleCreateModalClose}
462:                   className="w-full sm:w-auto px-4 py-2 text-gray-700 bg-gray-200 rounded-lg hover:bg-gray-
463:                 >
464:                   Cancel
465:                 </button>
466:                 <button
467:                   type="button"
468:                   onClick={addNewField}
469:                   className="w-full sm:w-auto px-4 py-2 bg-green-600 text-white rounded-lg hover:bg-green-7
470:                 >
471:                   Add Field
472:                 </button>
```

```
              </div>
            </div>
          </div>
        </div>
      )}
    </div>
  );
};

// ? Reusable components
const FieldInput = ({
  label,
  value,
  onChange,
  type = "text",
  disabled = false,
  placeholder = "",
  required = false,
  min,
}) => (
  <div className="flex flex-col">
    <label className="text-sm font-medium text-gray-700 mb-1">{label}</label>
    <input
      type={type}
      disabled={disabled}
      placeholder={placeholder}
      required={required}
      min={min}
      className={`w-full border rounded-md px-3 py-2 text-sm focus:outline-none focus:ring-2 focus:ring-blu
        disabled
          ? "bg-gray-100 text-gray-500 cursor-not-allowed"
          : "bg-white border-gray-300 hover:border-gray-400"
      }`}
      value={value ?? ""}
      onChange={(e) => onChange?.(e.target.value)}
    />
  </div>
);

const FieldTextarea = ({ label, value, onChange }) => (
  <div className="flex flex-col">
    <label className="text-sm font-medium text-gray-700 mb-1">{label}</label>
    <textarea
      rows={3}
      className="w-full border border-gray-300 rounded-md px-3 py-2 text-sm focus:outline-none focus:ring-2
      value={value ?? ""}
      onChange={(e) => onChange?.(e.target.value)}
    />
  </div>
);

const FieldSelect = ({ label, value, options, onChange }) => (
  <div className="flex flex-col">
    <label className="text-sm font-medium text-gray-700 mb-1">{label}</label>
    <select
      className="w-full border border-gray-300 rounded-md px-3 py-2 text-sm focus:outline-none focus:ring-2
      value={value ?? ""}
      onChange={(e) => onChange?.(e.target.value)}
    >
      <option value="">? None ?</option>
      {options.map((opt, i) => (
        <option key={i} value={opt}>
          {opt}
        </option>
      ))}
    </select>
  </div>
);

export default AdminPanel;
```

■ File: src\components\common\LoadingSpinner.js

```
================================================================================

--------------------------------------------------------------------------------
```

■ File: src\components\common\ResetAll.js

```
================================================================================
  1: import React from "react";
  2: import { toast } from "react-toastify";
  3:
  4: const ResetAll = () => {
  5:   const handleReset = () => {
  6:     const confirmReset = window.confirm(
  7:       "This will erase all unsaved changes and restore the app to its default state. Continue?"
  8:     );
  9:
 10:     if (!confirmReset) return;
 11:
 12:     // Clear form input, config, settings
 13:     localStorage.removeItem("genomics_form_data");
 14:     // Optionally remove other keys if added later (e.g., config, theme)
 15:     // localStorage.removeItem('genomics_config');
 16:
 17:     toast.success("App state reset. Reloading...");
 18:
 19:     setTimeout(() => {
 20:       window.location.reload(); // reload default from formConfig.json
 21:     }, 1000);
 22:   };
 23:
 24:   return (
 25:     <button
 26:       onClick={handleReset}
 27:       className="bg-red-600 hover:bg-red-700 text-white px-6 py-3 rounded-lg font-semibold"
 28:     >
 29:       ?? Reset All
 30:     </button>
 31:   );
 32: };
 33:
 34: export default ResetAll;

--------------------------------------------------------------------------------
```

■ File: src\components\common\TabNavigation.js

```
================================================================================
  1: import { useTheme } from "../../contexts/ThemeContext";
  2:
  3: const TabNavigation = ({ activeTab, setActiveTab }) => {
  4:   const { theme, toggleTheme } = useTheme();
  5:
  6:   const tabs = [
  7:     { id: "form", label: "? Form View", shortLabel: "? Form" },
  8:     { id: "admin", label: "?? Admin Panel", shortLabel: "?? Admin" },
  9:     { id: "settings", label: "?? Form Settings", shortLabel: "?? Settings" },
 10:   ];
 11:
 12:   return (
 13:     <div className="bg-white dark:bg-gray-800 shadow-sm border-b sticky top-0 z-10">
 14:       <div className="w-full px-2 sm:px-4 lg:px-6">
 15:         <div className="flex justify-between items-center py-2 sm:py-4">
 16:           {/* Tabs - Always start from left */}
 17:           <div className="flex gap-1 sm:gap-2 items-center">
 18:             {tabs.map((tab) => (
 19:               <button
 20:                 key={tab.id}
 21:                 role="tab"
 22:                 aria-selected={activeTab === tab.id}
 23:                 className={`px-2 sm:px-3 md:px-4 py-1.5 sm:py-2 rounded-md sm:rounded-lg font-medium text-x
 24:                   activeTab === tab.id
 25:                     ? "bg-green-600 text-white shadow-md"
 26:                     : "bg-gray-200 text-gray-800 hover:bg-gray-300 dark:bg-gray-700 dark:text-gray-100 dark
 27:                 }`}
```

```
                onClick={() => setActiveTab(tab.id)}
              >
                {/* Show short label on mobile, full label on larger screens */}
                <span className="sm:hidden">{tab.shortLabel}</span>
                <span className="hidden sm:inline">{tab.label}</span>
              </button>
            ))}
          </div>

          {/* Theme Toggle - Always on right */}
          <button
            onClick={toggleTheme}
            className="text-xs sm:text-sm px-2 sm:px-3 py-1.5 sm:py-2 bg-gray-100 dark:bg-gray-700 dark:tex
          >
            <span className="sm:hidden">{theme === "dark" ? "??" : "?"}</span>
            <span className="hidden sm:inline">
              {theme === "dark" ? "?? Light" : "? Dark"}
            </span>
          </button>
        </div>
      </div>
    </div>
  );
};

export default TabNavigation;
```

■ File: src\components\common\Toasts.js

```
import { ToastContainer } from "react-toastify";
import "react-toastify/dist/ReactToastify.css";

const Toasts = () => {
  return (
    <>
      <ToastContainer
        position="top-center"
        autoClose={2000}
        hideProgressBar={false}
        newestOnTop={true}
        closeOnClick
        pauseOnFocusLoss
        draggable
        pauseOnHover
        closeButton={false}
        theme="light"
        limit={3}
        toastClassName="!bg-white/95 !backdrop-blur-md !rounded-xl !shadow-lg !shadow-black/10 !border !bor
        bodyClassName="!p-0 !m-0"
        progressClassName="!bg-gradient-to-r !from-blue-500 !to-cyan-500 !h-1"
        style={{
          top: "20px",
          left: "50%",
          transform: "translateX(-50%)",
          width: "400px",
          maxWidth: "90vw",
        }}
      />

      <style jsx global>{`
        .Toastify__toast-container {
          @apply font-sans;
        }

        .Toastify__toast--success {
          @apply !bg-green-50/95 !border-l-4 !border-l-green-500 !text-green-800;
        }

        .Toastify__toast--error {
          @apply !bg-red-50/95 !border-l-4 !border-l-red-500 !text-red-800;
        }
```

```
43:
44:         .Toastify__toast--warning {
45:           @apply !bg-yellow-50/95 !border-l-4 !border-l-yellow-500 !text-yellow-800;
46:         }
47:
48:         .Toastify__toast--info {
49:           @apply !bg-blue-50/95 !border-l-4 !border-l-blue-500 !text-blue-800;
50:         }
51:       `}</style>
52:     </>
53:   );
54: };
55:
56: export default Toasts;
```

■ File: src\components\form\FieldInputRow.js

==========

■ File: src\components\form\InputForm.js

==========

```
  1: import React, { useState, useEffect } from "react";
  2: import { useExcelExport } from "../../hooks/useExcelExport";
  3: import { useFormConfig } from "../../contexts/FormConfigContext";
  4: import { usePDFGeneration } from "../../hooks/usePDFGeneration";
  5: import { isValidScore } from "../../utils/helpers";
  6: import { toast } from "react-toastify";
  7:
  8: const LOCAL_STORAGE_KEY = "genomics_form_data";
  9:
 10: const InputForm = ({ onGenerateReport, reportData }) => {
 11:   const { state } = useFormConfig();
 12:   const { generatePDF } = usePDFGeneration();
 13:
 14:   // Try restoring from localStorage
 15:   const [formData, setFormData] = useState(() => {
 16:     try {
 17:       const stored = localStorage.getItem(LOCAL_STORAGE_KEY);
 18:       return stored ? JSON.parse(stored) : {};
 19:     } catch (e) {
 20:       return {};
 21:     }
 22:   });
 23:
 24:   const { exportToExcel } = useExcelExport();
 25:
 26:   const [errors, setErrors] = useState({});
 27:
 28:   // ? Save to localStorage on formData change
 29:   useEffect(() => {
 30:     localStorage.setItem(LOCAL_STORAGE_KEY, JSON.stringify(formData));
 31:   }, [formData]);
 32:
 33:   const handleInputChange = (fieldId, value) => {
 34:     const updatedValue = value.replace(/\D/g, "");
 35:     setFormData((prev) => ({
 36:       ...prev,
 37:       [fieldId]: updatedValue,
 38:     }));
 39:     setErrors((prev) => ({
 40:       ...prev,
 41:       [fieldId]: null,
 42:     }));
 43:   };
 44:
 45:   const validateForm = () => {
 46:     const newErrors = {};
 47:     state.fields.forEach((field) => {
 48:       const value = formData[field.id];
 49:       if (!isValidScore(value)) {
```

```
        newErrors[field.id] = "Score must be between 1 and 10";
      }
    });
    setErrors(newErrors);
    return Object.keys(newErrors).length === 0;
  };

  const handleSubmit = (e) => {
    e.preventDefault();
    if (validateForm()) {
      onGenerateReport(formData);
      toast.success("Report generated and saved!");
    } else {
      toast.error("Please fix errors before submitting.");
    }
  };

  const downloadPDF = () => {
    if (!reportData || reportData.length === 0) {
      toast.warning("Please generate a report first.");
      return;
    }
    generatePDF(reportData);
  };

  const handleClearForm = () => {
    if (window.confirm("Clear all form scores and reset saved state?")) {
      localStorage.removeItem(LOCAL_STORAGE_KEY);
      setFormData({});
      toast.info("?? Cleared saved input.");
    }
  };

  return (
    <div className="w-full md:w-1/2 bg-white p-4 md:p-8 overflow-y-auto mobile-section">
      {/* ... header & description remain the same */}

      <form onSubmit={handleSubmit} className="space-y-3 md:space-y-4">
        <div className="grid grid-cols-1 gap-4">
          {state.fields.map((field, index) => (
            <div
              key={field.id}
              className="flex flex-col md:flex-row md:items-center"
            >
              <label className="w-full md:w-48 text-sm font-semibold mb-1 md:mb-0">
                {field.label}
              </label>
              <input
                type="number"
                min="1"
                max="10"
                value={formData[field.id] || ""}
                onChange={(e) => handleInputChange(field.id, e.target.value)}
                className={`border p-2 w-full md:w-20 text-center rounded
                  ${errors[field.id] ? "border-red-500" : "border-gray-300"}`}
              />
              {errors[field.id] && (
                <p className="text-red-600 text-xs mt-1 md:ml-4">
                  {errors[field.id]}
                </p>
              )}
            </div>
          ))}
        </div>

        <div className="flex flex-col md:flex-row gap-2 mt-6">
          <button
            type="submit"
            className="bg-green-600 hover:bg-green-700 text-white px-6 py-3 rounded-lg font-semibold"
          >
            Generate Report
          </button>

```

```
123:           <button
124:             type="button"
125:             onClick={handleClearForm}
126:             className="bg-gray-500 hover:bg-gray-600 text-white px-6 py-3 rounded-lg font-semibold"
127:           >
128:             Clear Input
129:           </button>
130:           <button
131:             type="button"
132:             onClick={() => exportToExcel(reportData)}
133:             className="bg-yellow-500 hover:bg-yellow-600 text-white px-6 py-3 rounded-lg font-semibold"
134:           >
135:             ? Export Excel
136:           </button>
137:         </div>
138:       </form>
139:     </div>
140:   );
141: };
142:
143: export default InputForm;
```

■ File: src\components\report\PDFPreview.js

==========

■ File: src\components\report\ReportOutput.js

==========

```
  1: import React, { useState, useEffect } from "react";
  2:
  3: const ReportOutput = ({ reportData }) => {
  4:   const [settings, setSettings] = useState(null);
  5:   const [leftLogoBase64, setLeftLogoBase64] = useState(null);
  6:   const [rightLogoBase64, setRightLogoBase64] = useState(2);
  7:
  8:   // Fetch settings from API
  9:   useEffect(() => {
 10:     const fetchSettings = async () => {
 11:       try {
 12:         const res = await fetch("http://localhost:5000/settings");
 13:         const data = await res.json();
 14:         setSettings(data);
 15:       } catch (error) {
 16:         console.error("Failed to fetch settings:", error);
 17:       }
 18:     };
 19:
 20:     fetchSettings();
 21:   }, []);
 22:
 23:   // Handle loading state
 24:   if (!settings) return <div>Loading report...</div>;
 25:
 26:   // Assign color to score
 27:   const getScoreColor = (score) => {
 28:     if (score >= settings.highThreshold) return "bg-red-600";
 29:     if (score >= 4) return "bg-yellow-500";
 30:     return "bg-green-600";
 31:   };
 32:
 33:   // Style active/inactive text
 34:   const getTextStyle = (isActive) =>
 35:     isActive ? "text-gray-900 font-semibold" : "text-gray-400";
 36:
 37:   // Helper function to convert hex to RGB
 38:   const hexToRgb = (hex) => {
 39:     const result = /^#?([a-f\d]{2})([a-f\d]{2})([a-f\d]{2})$/i.exec(hex);
 40:     return result
 41:       ? [
 42:           parseInt(result[1], 16),
```

```
          parseInt(result[2], 16),
          parseInt(result[3], 16),
        ]
      : [0, 0, 0];
  };

  // PDF Download functionality
  const downloadPDF = async () => {
    if (!reportData || reportData.length === 0) {
      alert("Please generate a report first by filling in all the scores.");
      return;
    }

    // Dynamically import jsPDF
    const { jsPDF } = await import("jspdf");
    const doc = new jsPDF();

    const pageHeight = doc.internal.pageSize.height; // 297mm for A4
    const pageWidth = doc.internal.pageSize.width; // 210mm for A4

    // Function to add logos to each page
    const addLogosToPage = () => {
      // Left logo
      if (leftLogoBase64) {
        try {
          doc.addImage(leftLogoBase64, "PNG", 10, pageHeight - 25, 30, 10);
        } catch (e) {
          console.warn("Could not add left logo:", e);
        }
      }
      // Right logo
      if (rightLogoBase64) {
        try {
          doc.addImage(
            rightLogoBase64,
            "PNG",
            pageWidth - 40,
            pageHeight - 25,
            30,
            10
          );
        } catch (e) {
          console.warn("Could not add right logo:", e);
        }
      }
    };

    // Header
    doc.setFillColor(...hexToRgb(settings.headerColor || "#16a34a"));
    doc.rect(10, 10, pageWidth - 20, 20, "F");
    doc.setTextColor(255, 255, 255);
    doc.setFontSize(18);
    doc.setFont(undefined, "bold");
    doc.text(settings.title || "GENOMICS & DIET", pageWidth / 2, 22, {
      align: "center",
    });

    // Quote and description
    doc.setTextColor(0, 0, 0);
    doc.setFontSize(12);
    doc.setFont(undefined, "bold");
    doc.text(
      settings.quote || '"YOU ARE WHAT YOU EAT" - Victor Lindlahr',
      pageWidth / 2,
      40,
      { align: "center" }
    );

    doc.setFontSize(10);
    doc.setFont(undefined, "normal");
    const splitDescription = doc.splitTextToSize(
      settings.description || "",
      pageWidth - 30
```

```
116:     );
117:     doc.text(splitDescription, 15, 50);
118:
119:     // Add logos to first page
120:     addLogosToPage();
121:
122:     let yPosition = 65 + splitDescription.length * 4;
123:     let currentCategory = "";
124:
125:     // Generate report content
126:     reportData.forEach((item, index) => {
127:       const { field, score } = item;
128:       const showHigh = score >= settings.highThreshold;
129:       const showNormal = score >= 4 && score < settings.highThreshold;
130:       const showLow = score < 4;
131:
132:       // Add category header if needed
133:       if (field.category && field.category !== currentCategory) {
134:         currentCategory = field.category;
135:
136:         // Check if we need a new page for category header
137:         if (yPosition > pageHeight - 50) {
138:           doc.addPage();
139:           addLogosToPage();
140:           yPosition = 20;
141:         }
142:
143:         doc.setFillColor(220, 220, 220);
144:         doc.rect(10, yPosition, pageWidth - 20, 8, "F");
145:         doc.setTextColor(0, 0, 0);
146:         doc.setFontSize(10);
147:         doc.setFont(undefined, "bold");
148:         doc.text(currentCategory, 15, yPosition + 5);
149:         yPosition += 12;
150:       }
151:
152:       // Estimate content height before adding
153:       const tempHighText = doc.splitTextToSize(
154:         field.high.replace(/\n/g, " "),
155:         (pageWidth - 40) / 5 - 5
156:       );
157:       const tempNormalText = doc.splitTextToSize(
158:         field.normal ? field.normal.replace(/\n/g, " ") : "",
159:         (pageWidth - 40) / 5 - 5
160:       );
161:       const tempLowText = doc.splitTextToSize(
162:         field.low.replace(/\n/g, " "),
163:         (pageWidth - 40) / 5 - 5
164:       );
165:
166:       const estimatedHeight =
167:         Math.max(
168:           tempHighText.length,
169:           tempNormalText.length,
170:           tempLowText.length,
171:           1
172:         ) *
173:           3.5 +
174:         15;
175:
176:       // Check if we need a new page
177:       if (yPosition + estimatedHeight > pageHeight - 15) {
178:         doc.addPage();
179:         addLogosToPage();
180:         yPosition = 20;
181:       }
182:
183:       // Dynamic column layout based on page width
184:       const leftMargin = 10;
185:       const rightMargin = 10;
186:       const usableWidth = pageWidth - leftMargin - rightMargin;
187:       const columnSpacing = usableWidth / 5;
188:
```

```
189:       // Calculate positions for equal spacing
190:       const fieldLabelX = leftMargin;
191:       const scoreCircleX = leftMargin + columnSpacing * 1.5;
192:       const highX = leftMargin + columnSpacing * 2;
193:       const normalX = leftMargin + columnSpacing * 3;
194:       const lowX = leftMargin + columnSpacing * 4;
195:
196:       const colWidth = columnSpacing - 3;
197:
198:       // Field label and score section
199:       doc.setFontSize(9);
200:       doc.setFont(undefined, "bold");
201:       doc.setTextColor(0, 0, 0);
202:
203:       // Field label (first column)
204:       doc.text(field.label, fieldLabelX, yPosition + 4);
205:
206:       // Score circle (second column)
207:       const scoreColor =
208:         score >= settings.highThreshold
209:           ? hexToRgb(settings.colors.high)
210:           : score >= 4
211:           ? hexToRgb(settings.colors.medium)
212:           : hexToRgb(settings.colors.low);
213:       doc.setFillColor(...scoreColor);
214:       const circleY = yPosition + 4;
215:       doc.circle(scoreCircleX, circleY, 4, "F");
216:
217:       // Score number in circle
218:       doc.setTextColor(255, 255, 255);
219:       doc.setFontSize(10);
220:       doc.setFont(undefined, "bold");
221:       doc.text(score.toString(), scoreCircleX, circleY + 1.5, {
222:         align: "center",
223:       });
224:
225:       // Reset text color for recommendations
226:       doc.setTextColor(0, 0, 0);
227:       doc.setFontSize(8);
228:
229:       // HIGH column (third column)
230:       const highColor = showHigh ? [0, 0, 0] : [156, 163, 175];
231:       doc.setTextColor(...highColor);
232:       doc.setFont(undefined, "bold");
233:       doc.text("HIGH", highX, yPosition + 2);
234:       doc.setFont(undefined, "normal");
235:       const highText = doc.splitTextToSize(
236:         field.high.replace(/\n/g, " "),
237:         colWidth
238:       );
239:       doc.text(highText, highX, yPosition + 6);
240:
241:       // NORMAL column (fourth column)
242:       const normalColor = showNormal ? [0, 0, 0] : [156, 163, 175];
243:       doc.setTextColor(...normalColor);
244:       doc.setFont(undefined, "bold");
245:       doc.text("NORMAL", normalX, yPosition + 2);
246:       doc.setFont(undefined, "normal");
247:       const normalText = doc.splitTextToSize(
248:         field.normal ? field.normal.replace(/\n/g, " ") : "",
249:         colWidth
250:       );
251:       doc.text(normalText, normalX, yPosition + 6);
252:
253:       // LOW column (fifth column)
254:       const lowColor = showLow ? [0, 0, 0] : [156, 163, 175];
255:       doc.setTextColor(...lowColor);
256:       doc.setFont(undefined, "bold");
257:       doc.text("LOW", lowX, yPosition + 2);
258:       doc.setFont(undefined, "normal");
259:       const lowText = doc.splitTextToSize(
260:         field.low.replace(/\n/g, " "),
261:         colWidth
```

```
      );
      doc.text(lowText, lowX, yPosition + 6);

      // Calculate next Y position based on the tallest column
      const maxLines = Math.max(
        highText.length,
        normalText.length,
        lowText.length,
        1
      );
      yPosition += maxLines * 3.2 + 10;

      // Add a thin separator line (but not after the last item)
      if (index < reportData.length - 1) {
        doc.setDrawColor(200, 200, 200);
        doc.setLineWidth(0.2);
        doc.line(10, yPosition - 4, pageWidth - 10, yPosition - 4);
      }
    });

    // Save the PDF
    doc.save("genomics-diet-report.pdf");
  };

  if (!reportData || reportData.length === 0) {
    return (
      <div className="w-full md:w-1/2 p-4 md:p-8 mobile-section bg-gray-50">
        <div className="bg-white border border-gray-300 rounded-lg p-8 text-center text-gray-500">
          Fill in the form and generate report to view results here.
        </div>
      </div>
    );
  }

  let currentCategory = null;

  return (
    <div className="w-full md:w-1/2 p-4 md:p-8 mobile-section bg-gray-50 overflow-y-auto">
      <div className="bg-white border border-gray-300 rounded-lg shadow-sm overflow-hidden">
        {/* PDF Download Button */}
        <div className="p-4 bg-gray-50 border-b border-gray-200">
          <button
            onClick={downloadPDF}
            className="bg-blue-600 hover:bg-blue-700 text-white px-4 py-2 rounded-lg font-semibizer transit
          >
            Download PDF Report
          </button>
        </div>

        {reportData.map((item, index) => {
          const { field, score, showHigh, showNormal, showLow } = item;
          const isNewCategory =
            field.category && field.category !== currentCategory;
          const elements = [];

          // Add category header if changed
          if (isNewCategory) {
            currentCategory = field.category;
            elements.push(
              <div
                key={`category-${field.category}`}
                className="bg-gray-200 px-4 py-2 font-bold text-sm text-gray-700 border-l-4 border-gray-400
              >
                {currentCategory}
              </div>
            );
          }

          // Add field row
          elements.push(
            <div
              key={field.id}
              className="flex flex-col md:flex-row items-stretch border-b border-gray-200"
```

```
            >
              {/* Field Label */}
              <div className="w-full md:w-48 px-3 py-3 text-center md:text-right bg-gray-100 md:bg-white">
                <div className="text-xs font-bold text-gray-700 uppercase leading-tight">
                  {field.label}
                </div>
              </div>

              {/* Score */}
              <div className="w-full md:w-16 flex justify-center items-center py-3">
                <div
                  className={`w-10 h-10 ${getScoreColor(
                    score
                  )} text-white font-bold text-lg flex items-center justify-center rounded-full`}
                >
                  {score}
                </div>
              </div>

              {/* Recommendations */}
              <div className="flex-1 px-3 py-3 flex flex-col md:flex-row">
                {/* High */}
                <div className="w-full md:w-1/3 md:pr-2 mb-3 md:mb-0">
                  <div
                    className={`text-xs font-bold mb-1 ${getTextStyle(
                      showHigh
                    )}`}
                  >
                    HIGH
                  </div>
                  <div
                    className={`text-xs leading-tight ${getTextStyle(
                      showHigh
                    )}`}
                  >
                    {field.high.split("\n").map((line, i, arr) => (
                      <React.Fragment key={i}>
                        {line}
                        {i < arr.length - 1 && <br />}
                      </React.Fragment>
                    ))}
                  </div>
                </div>

                {/* Normal */}
                <div className="w-full md:w-1/3 md:px-2 mb-3 md:mb-0">
                  <div
                    className={`text-xs font-bold mb-1 ${getTextStyle(
                      showNormal
                    )}`}
                  >
                    NORMAL
                  </div>
                  <div
                    className={`text-ys leading-tight ${getTextStyle(
                      showNormal
                    )}`}
                  >
                    {field.normal?.split("\n").map((line, i, arr) => (
                      <React.Fragment key={i}>
                        {line}
                        {i < arr.length - 1 && <br />}
                      </React.Fragment>
                    ))}
                  </div>
                </div>

                {/* Low */}
                <div className="w-full md:w-1/3 md:pl-2">
                  <div
                    className={`text-xs font-bold mb-1 ${getTextStyle(
                      showLow
                    )}`}
```

```
408:                   >
409:                     LOW
410:                   </div>
411:                   <div
412:                     className={`text-xs leading-tight ${getTextStyle(showLow)}`}
413:                   >
414:                     {field.low.split("\n").map((line, i, arr) => (
415:                       <React.Fragment key={i}>
416:                         {line}
417:                         {i < arr.length - 1 && <br />}
418:                       </React.Fragment>
419:                     ))}
420:                   </div>
421:                 </div>
422:               </div>
423:             </div>
424:           );
425:
426:           return elements;
427:         })}
428:       </div>
429:     </div>
430:   );
431: };
432:
433: export default ReportOutput;
```

■ File: src\components\settings\CategoryManager.js

```
  1: import { useState, useEffect } from "react";
  2: import { toast } from "react-toastify";
  3:
  4: const API_URL = "http://localhost:5000";
  5:
  6: const CategoryManager = () => {
  7:   const [categories, setCategories] = useState([]);
  8:   const [newCategory, setNewCategory] = useState("");
  9:   const [editIndex, setEditIndex] = useState(null);
 10:   const [editedName, setEditedName] = useState("");
 11:
 12:   // ? Fetch from server
 13:   const fetchCategories = async () => {
 14:     try {
 15:       const res = await fetch(`${API_URL}/categories`);
 16:       const data = await res.json();
 17:       setCategories(data);
 18:     } catch (err) {
 19:       toast.error("Failed to load categories");
 20:     }
 21:   };
 22:
 23:   useEffect(() => {
 24:     fetchCategories();
 25:   }, []);
 26:
 27:   // ? Add
 28:   const addCategory = async () => {
 29:     const trimmed = newCategory.trim();
 30:     if (!trimmed) return;
 31:
 32:     const exists = categories.some((c) => c.name === trimmed);
 33:     if (exists) {
 34:       toast.error("Category already exists");
 35:       return;
 36:     }
 37:
 38:     const res = await fetch(`${API_URL}/categories`, {
 39:       method: "POST",
 40:       headers: { "Content-Type": "application/json" },
 41:       body: JSON.stringify({ name: trimmed }),
 42:     });
```

```

    if (res.ok) {
      toast.success("Category added");
      setNewCategory("");
      fetchCategories(); // ? refresh list
    }
  };

  // ?? Start editing
  const startEdit = (i, name) => {
    setEditIndex(i);
    setEditedName(name);
  };

  // ? Confirm edit
  const confirmEdit = async () => {
    if (!editedName.trim()) return;

    const cat = categories[editIndex];

    const res = await fetch(`${API_URL}/categories/${cat.id}`, {
      method: "PUT",
      headers: { "Content-Type": "application/json" },
      body: JSON.stringify({ ...cat, name: editedName.trim() }),
    });

    if (res.ok) {
      toast.success("Category updated");
      setEditIndex(null);
      setEditedName("");
      fetchCategories(); // ? refresh
    } else {
      toast.error("? Failed to update category");
    }
  };

  // ?? Delete
  const deleteCategory = async (index) => {
    const cat = categories[index];

    if (!window.confirm(`Delete category "${cat.name}"?`)) return;

    const res = await fetch(`${API_URL}/categories/${cat.id}`, {
      method: "DELETE",
    });

    if (res.ok) {
      toast.info("Category deleted");
      fetchCategories(); // ? refresh
    } else {
      toast.error("Failed to delete category");
    }
  };

  return (
    <div className="mt-10 border-t pt-6">
      <h3 className="text-xl font-semibold mb-4">Category Manager</h3>

      <div className="flex gap-2 mb-4">
        <input
          type="text"
          className="border px-3 py-2 rounded w-full"
          placeholder="New category name"
          value={newCategory}
          onChange={(e) => setNewCategory(e.target.value)}
        />
        <button
          onClick={addCategory}
          className="bg-green-600 hover:bg-green-700 text-white px-4 py-2 rounded"
        >
          Add
        </button>
      </div>
```

```
116:
117:       <div className="space-y-3">
118:         {categories.map((cat, i) => (
119:           <div
120:             key={cat.id}
121:             className="flex items-center justify-between bg-white border p-3 rounded"
122:           >
123:             {editIndex === i ? (
124:               <>
125:                 <input
126:                   type="text"
127:                   className="border px-2 py-1 rounded w-full mr-2"
128:                   value={editedName}
129:                   onChange={(e) => setEditedName(e.target.value)}
130:                 />
131:                 <button
132:                   onClick={confirmEdit}
133:                   className="bg-blue-600 text-white p-2 rounded"
134:                 >Save</button>
135:               </>
136:             ) : (
137:               <>
138:                 <span>{cat.name}</span>
139:
140:                 <div className="flex gap-2">
141:                   <button
142:                     onClick={() => startEdit(i, cat.name)}
143:                     className="inline-flex items-center px-3 py-1 bg-blue-600 text-white text-sm rounded-md
144:                   >
145:                     Edit
146:                   </button>
147:                   <button
148:                     onClick={() => deleteCategory(i)}
149:                     className="inline-flex items-center px-3 py-1 bg-red-600 text-white text-sm rounded-md
150:                   >
151:                     Delete
152:                   </button>
153:                 </div>
154:               </>
155:             )}
156:           </div>
157:         ))}
158:       </div>
159:     </div>
160:   );
161: };
162:
163: export default CategoryManager;
```

■ File: src\components\settings\SettingsPanel.js

```
  1: import React, { useState, useRef } from "react";
  2: import { useFormConfig } from "../../contexts/FormConfigContext";
  3: import { useConfigImportExport } from "../../hooks/useConfigImportExport";
  4: import { toast } from "react-toastify";
  5: import ResetAll from "../common/ResetAll";
  6: import CategoryManager from "./CategoryManager";
  7:
  8: const API_URL = "http://localhost:5000";
  9:
 10: const SettingsPanel = () => {
 11:   const { state, dispatch } = useFormConfig();
 12:
 13:   const [settings, setSettings] = useState({
 14:     title: state.title,
 15:     quote: state.quote,
 16:     description: state.description,
 17:     headerColor: state.headerColor,
 18:     highThreshold: state.highThreshold,
 19:     colors: {
 20:       low: state.colors.low,
```

```
      medium: state.colors.medium,
      high: state.colors.high,
    },
  });

  const fileInputRef = useRef(null);
  const { exportConfig, importConfig } = useConfigImportExport();

  const handleImportClick = () => {
    fileInputRef.current?.click();
  };

  const handleFileSelected = (e) => {
    const file = e.target.files?.[0];
    if (!file || !file.name.endsWith(".json")) {
      toast.error("Please upload a valid JSON file.");
      return;
    }
    importConfig(file);
    e.target.value = ""; // reset input
  };

  // ? Update local state as user types
  const handleChange = (key, value) => {
    setSettings((prev) => ({
      ...prev,
      [key]: value,
    }));
  };

  const handleColorChange = (level, value) => {
    setSettings((prev) => ({
      ...prev,
      colors: {
        ...prev.colors,
        [level]: value,
      },
    }));
  };

  // ? Save to backend
  const applySettings = async () => {
    try {
      const res = await fetch(`${API_URL}/settings`, {
        method: "PUT",
        headers: { "Content-Type": "application/json" },
        body: JSON.stringify(settings),
      });

      if (!res.ok) throw new Error("Failed to save settings");

      dispatch({ type: "UPDATE_SETTINGS", settings }); // update UI too
      toast.success("Settings saved to server!");
    } catch (error) {
      console.error(error);
      toast.error("Failed to save settings");
    }
  };

  const resetSettings = () => {
    if (
      window.confirm("Are you sure you want to reset all settings to default?")
    ) {
      window.location.reload(); // simplest way
    }
  };

  return (
    <div className="max-w-full mx-auto px-4 py-8">
      <h2 className=" bg-white text-2xl font-bold mb-6">
        ?? Form Customization
      </h2>

```

```
 94:       <div className="grid grid-cols-1 md:grid-cols-2 gap-6">
 95:         {/* Left Section */}
 96:         <div className="bg-white rounded-lg shadow-sm p-6 space-y-4">
 97:           <h3 className="text-lg font-semibold">? Header Info</h3>
 98:
 99:           <div>
100:             <label className="block text-sm font-medium mb-1">Main Title</label>
101:             <input
102:               type="text"
103:               value={settings.title}
104:               onChange={(e) => handleChange("title", e.target.value)}
105:               className="w-full border rounded px-3 py-2"
106:             />
107:           </div>
108:
109:           <div>
110:             <label className="block text-sm font-medium mb-1">Quote</label>
111:             <textarea
112:               rows={2}
113:               value={settings.quote}
114:               onChange={(e) => handleChange("quote", e.target.value)}
115:               className="w-full border rounded px-3 py-2"
116:             />
117:           </div>
118:
119:           <div>
120:             <label className="block text-sm font-medium mb-1">
121:               Description
122:             </label>
123:             <textarea
124:               rows={4}
125:               value={settings.description}
126:               onChange={(e) => handleChange("description", e.target.value)}
127:               className="w-full border rounded px-3 py-2"
128:             />
129:           </div>
130:
131:           <div>
132:             <label className="block text-sm font-medium mb-1">
133:               Header Background Color
134:             </label>
135:             <input
136:               type="color"
137:               value={settings.headerColor}
138:               onChange={(e) => handleChange("headerColor", e.target.value)}
139:               className="h-10 w-full border rounded"
140:             />
141:           </div>
142:         </div>
143:
144:         {/* Right Section */}
145:         <div className="bg-white rounded-lg shadow-sm p-6 space-y-4">
146:           <h3 className="text-lg font-semibold">? Score Logic</h3>
147:
148:           <div>
149:             <label className="block text-sm font-medium mb-1">
150:               High Score Threshold (?)
151:             </label>
152:             <input
153:               type="number"
154:               min="1"
155:               max="10"
156:               value={settings.highThreshold}
157:               onChange={(e) =>
158:                 handleChange("highThreshold", parseInt(e.target.value))
159:               }
160:               className="w-full border rounded px-3 py-2"
161:             />
162:           </div>
163:
164:           <h3 className="text-lg font-semibold mt-6">? Score Colors</h3>
165:
166:           <div className="space-y-2">
```

```
            <ColorInput
              label="High"
              value={settings.colors.high}
              onChange={(val) => handleColorChange("high", val)}
            />
            <ColorInput
              label="Medium"
              value={settings.colors.medium}
              onChange={(val) => handleColorChange("medium", val)}
            />
            <ColorInput
              label="Low"
              value={settings.colors.low}
              onChange={(val) => handleColorChange("low", val)}
            />
          </div>
        </div>
      </div>

      {/* Buttons */}
      <div className="mt-6 flex flex-wrap gap-4">
        <button
          onClick={applySettings}
          className="bg-green-600 hover:bg-green-700 text-white px-6 py-3 rounded-lg font-semibold"
        >
          ? Apply Settings
        </button>
        <button
          onClick={resetSettings}
          className="bg-gray-600 hover:bg-gray-700 text-white px-6 py-3 rounded-lg font-semibold"
        >
          ?? Reset to Default
        </button>
        <button
          onClick={exportConfig}
          className="bg-blue-600 hover:bg-blue-700 text-white px-6 py-3 rounded-lg font-semibold"
        >
          ? Export Config (.json)
        </button>

        <button
          onClick={handleImportClick}
          className="bg-purple-600 hover:bg-purple-700 text-white px-6 py-3 rounded-lg font-semibold"
        >
          ? Import Config (.json)
        </button>

        <div className="mt-2">
          <ResetAll />
        </div>

        <input
          type="file"
          accept=".json"
          ref={fileInputRef}
          className="hidden"
          onChange={handleFileSelected}
        />
      </div>

      {/* ?? Category management (uses API now) */}
      <div className="mt-6">
        <CategoryManager />
      </div>
    </div>
  );
};

const ColorInput = ({ label, value, onChange }) => (
  <div>
    <label className="block text-sm font-medium mb-1">{label}</label>
    <input
      type="color"
```

```
240:       value={value}
241:       onChange={(e) => onChange(e.target.value)}
242:       className="h-10 w-full border rounded"
243:     />
244:   </div>
245: );
246:
247: export default SettingsPanel;
```

■ File: src\contexts\FormConfigContext.js

```
==========================================================================
  1: import React, { createContext, useContext, useReducer, useEffect } from "react";
  2:
  3: // Base URL of your json-server
  4: const API_URL = "http://localhost:5000";
  5:
  6: const formConfigReducer = (state, action) => {
  7:   switch (action.type) {
  8:     case "IMPORT_CONFIG":
  9:       return { ...action.config };
 10:
 11:     case "ADD_FIELD":
 12:       return { ...state, fields: [...state.fields, action.field] };
 13:
 14:     case "UPDATE_FIELD":
 15:       if (action.property === "full") {
 16:         return {
 17:           ...state,
 18:           fields: state.fields.map((field, i) =>
 19:             i === action.index ? action.value : field
 20:           ),
 21:         };
 22:       }
 23:       return {
 24:         ...state,
 25:         fields: state.fields.map((field, i) =>
 26:           i === action.index
 27:             ? { ...field, [action.property]: action.value }
 28:             : field
 29:         ),
 30:       };
 31:
 32:     case "DELETE_FIELD":
 33:       return {
 34:         ...state,
 35:         fields: state.fields.filter((_, i) => i !== action.index),
 36:       };
 37:
 38:     case "REORDER_FIELDS":
 39:       return { ...state, fields: action.fields };
 40:
 41:     case "UPDATE_SETTINGS":
 42:       return { ...state, ...action.settings };
 43:
 44:     case "ADD_CATEGORY":
 45:       return {
 46:         ...state,
 47:         categories: [
 48:           ...state.categories,
 49:           { id: Date.now(), name: action.name },
 50:         ],
 51:       };
 52:
 53:     case "UPDATE_CATEGORY":
 54:       return {
 55:         ...state,
 56:         categories: state.categories.map((cat, i) =>
 57:           i === action.index ? { ...cat, name: action.newName } : cat
 58:         ),
 59:         fields: state.fields.map((field) =>
 60:           field.category === state.categories[action.index]?.name
```

```
            ? { ...field, category: action.newName }
            : field
        ),
      };

    case "DELETE_CATEGORY":
      const catName = state.categories[action.index]?.name;
      return {
        ...state,
        categories: state.categories.filter((_, i) => i !== action.index),
        fields: state.fields.map((field) =>
          field.category === catName ? { ...field, category: "" } : field
        ),
      };

    default:
      return state;
  }
};

const FormConfigContext = createContext(null);

export const FormConfigProvider = ({ children }) => {
  const [state, dispatch] = useReducer(formConfigReducer, {
    title: "",
    quote: "",
    description: "",
    headerColor: "",
    colors: { low: "", medium: "", high: "" },
    highThreshold: 6,
    categories: [],
    fields: [],
  });

  // ? Load config from json-server at startup
  useEffect(() => {
    const loadFromServer = async () => {
      try {
        const [settingsRes, categoriesRes, fieldsRes] = await Promise.all([
          fetch(`${API_URL}/settings`),
          fetch(`${API_URL}/categories`),
          fetch(`${API_URL}/fields`),
        ]);

        const settings = await settingsRes.json();
        const categories = await categoriesRes.json();
        const fields = await fieldsRes.json();

        dispatch({
          type: "IMPORT_CONFIG",
          config: {
            ...settings,
            categories, // array of { id, name }
            fields, // array of full field objects
          },
        });
      } catch (err) {
        console.error("Failed to fetch config from API", err);
      }
    };

    loadFromServer();
  }, []);

  return (
    <FormConfigContext.Provider value={{ state, dispatch }}>
      {children}
    </FormConfigContext.Provider>
  );
};

export const useFormConfig = () => {
  const context = useContext(FormConfigContext);
```

```
134:   if (!context)
135:     throw new Error("useFormConfig must be used within a FormConfigProvider");
136:   return context;
137: };
```

■ File: src\contexts\ThemeContext.js

```
  1: import React, { createContext, useEffect, useState, useContext } from "react";
  2:
  3: const ThemeContext = createContext();
  4:
  5: export const ThemeProvider = ({ children }) => {
  6:   const [theme, setTheme] = useState("light");
  7:
  8:   useEffect(() => {
  9:     const stored = localStorage.getItem("theme");
 10:     if (stored === "dark") {
 11:       document.documentElement.classList.add("dark");
 12:       setTheme("dark");
 13:     }
 14:   }, []);
 15:
 16:   const toggleTheme = () => {
 17:     const nextTheme = theme === "dark" ? "light" : "dark";
 18:     setTheme(nextTheme);
 19:     localStorage.setItem("theme", nextTheme);
 20:     document.documentElement.classList.toggle("dark");
 21:   };
 22:
 23:   return (
 24:     <ThemeContext.Provider value={{ theme, toggleTheme }}>
 25:       {children}
 26:     </ThemeContext.Provider>
 27:   );
 28: };
 29:
 30: export const useTheme = () => useContext(ThemeContext);
```

■ File: src\hooks\useConfigImportExport.js

```
  1: import { useCallback } from "react";
  2: import { useFormConfig } from "../contexts/FormConfigContext";
  3:
  4: const API_URL = "http://localhost:5000";
  5:
  6: export const useConfigImportExport = () => {
  7:   const { state, dispatch } = useFormConfig();
  8:
  9:   // ?? Export config from current state to file
 10:   const exportConfig = useCallback(() => {
 11:     const dataStr = JSON.stringify(state, null, 2);
 12:     const blob = new Blob([dataStr], { type: "application/json" });
 13:     const url = URL.createObjectURL(blob);
 14:
 15:     const link = document.createElement("a");
 16:     link.href = url;
 17:     link.download = "genomics-form-config.json";
 18:     link.click();
 19:     URL.revokeObjectURL(url);
 20:   }, [state]);
 21:
 22:   // ?? Import config and write to all backend endpoints
 23:   const importConfig = useCallback(
 24:     async (file) => {
 25:       const reader = new FileReader();
 26:
 27:       reader.onload = async (e) => {
 28:         try {
 29:           const parsed = JSON.parse(e.target.result);
```

```

          // Validate structure
          if (
            !parsed.fields ||
            !parsed.categories ||
            !parsed.title ||
            !parsed.colors
          ) {
            alert("Invalid configuration file.");
            return;
          }

          // ?? Overwrite ALL current data via PUT/DELETE/POST
          await Promise.all([
            // Clear old fields
            fetch(`${API_URL}/fields`)
              .then((res) => res.json())
              .then((existing) =>
                Promise.all(
                  existing.map((f) =>
                    fetch(`${API_URL}/fields/${f.id}`, { method: "DELETE" })
                  )
                )
              ),

            // Clear old categories
            fetch(`${API_URL}/categories`)
              .then((res) => res.json())
              .then((existing) =>
                Promise.all(
                  existing.map((c) =>
                    fetch(`${API_URL}/categories/${c.id}`, { method: "DELETE" })
                  )
                )
              ),
          ]);

          // ? Upload settings
          await fetch(`${API_URL}/settings`, {
            method: "PUT",
            headers: { "Content-Type": "application/json" },
            body: JSON.stringify({
              title: parsed.title,
              quote: parsed.quote,
              description: parsed.description,
              headerColor: parsed.headerColor,
              colors: parsed.colors,
              highThreshold: parsed.highThreshold,
            }),
          });

          // ? Upload categories
          for (const cat of parsed.categories) {
            await fetch(`${API_URL}/categories`, {
              method: "POST",
              headers: { "Content-Type": "application/json" },
              body: JSON.stringify(
                typeof cat === "string" ? { name: cat } : cat
              ),
            });
          }

          // ? Upload fields
          for (const field of parsed.fields) {
            await fetch(`${API_URL}/fields`, {
              method: "POST",
              headers: { "Content-Type": "application/json" },
              body: JSON.stringify(field),
            });
          }

          // ? Dispatch to update UI immediately
          dispatch({ type: "IMPORT_CONFIG", config: parsed });
```

```
103:           alert("? Configuration imported and saved to server.");
104:         } catch (err) {
105:           console.error(err);
106:           alert("? Error importing config: " + err.message);
107:         }
108:       };
109:
110:       reader.readAsText(file);
111:     },
112:     [dispatch]
113:   );
114:
115:   return { exportConfig, importConfig };
116: };
```

■ File: src\hooks\useExcelExport.js

```
  1: import * as XLSX from "xlsx";
  2:
  3: export const useExcelExport = () => {
  4:   const exportToExcel = (reportData) => {
  5:     if (!reportData || reportData.length === 0) {
  6:       alert("No report data to export.");
  7:       return;
  8:     }
  9:
 10:     const exportRows = reportData.map((item) => {
 11:       const { field, score, showHigh, showNormal, showLow } = item;
 12:
 13:       let recommendation = "";
 14:       if (showHigh) recommendation = field.high;
 15:       else if (showNormal) recommendation = field.normal;
 16:       else if (showLow) recommendation = field.low;
 17:
 18:       return {
 19:         Field: field.label,
 20:         Category: field.category || "Uncategorized",
 21:         Score: score,
 22:         Recommendation: recommendation.replace(/\n/g, " "),
 23:       };
 24:     });
 25:
 26:     const worksheet = XLSX.utils.json_to_sheet(exportRows);
 27:     const workbook = XLSX.utils.book_new();
 28:     XLSX.utils.book_append_sheet(workbook, worksheet, "Genomics Report");
 29:
 30:     XLSX.writeFile(workbook, "genomics_diet_report.xlsx");
 31:   };
 32:
 33:   return { exportToExcel };
 34: };
```

■ File: src\hooks\usePDFGeneration.js

```
  1: import { useCallback } from "react";
  2: import { jsPDF } from "jspdf";
  3: import { useFormConfig } from "../contexts/FormConfigContext";
  4: import { hexToRgb } from "../utils/helpers";
  5:
  6: export const usePDFGeneration = () => {
  7:   const { state } = useFormConfig();
  8:
  9:   // Base64 logo (optional)
 10:   const leftLogoUrl = "/left.png";
 11:   const rightLogoUrl = "/right.png";
 12:
 13:   const generatePDF = useCallback(
 14:     (reportData) => {
 15:       if (!reportData || reportData.length === 0) {
```

```
        alert("No report data found.");
        return;
      }

      const doc = new jsPDF();
      const pageHeight = doc.internal.pageSize.height;
      const pageWidth = doc.internal.pageSize.width;
      const margin = 10;
      let y = 20;

      // ----------------------------
      // Header section
      // ----------------------------
      doc.setFillColor(...hexToRgb(state.headerColor));
      doc.rect(margin, y, pageWidth - margin * 2, 20, "F");
      doc.setTextColor(255, 255, 255);
      doc.setFontSize(16);
      doc.setFont(undefined, "bold");
      doc.text(state.title, pageWidth / 2, y + 13, { align: "center" });
      y += 30;

      // Quote
      doc.setTextColor(0, 0, 0);
      doc.setFontSize(11);
      doc.setFont(undefined, "bold");
      doc.text(state.quote, pageWidth / 2, y, { align: "center" });
      y += 10;

      // Description
      doc.setFont(undefined, "normal");
      doc.setFontSize(10);
      const descLines = doc.splitTextToSize(
        state.description,
        pageWidth - 2 * margin
      );
      doc.text(descLines, margin, y);
      y += descLines.length * 5 + 5;

      let currentCategory = null;

      reportData.forEach((item, index) => {
        const { field, score, showHigh, showNormal, showLow } = item;

        // Insert page break if needed
        const estimatedFieldHeight = 40; // Rough estimate
        if (y + estimatedFieldHeight > pageHeight - 20) {
          doc.addPage();
          y = 20;
        }

        // Render category title if needed
        if (field.category && field.category !== currentCategory) {
          currentCategory = field.category;
          doc.setFontSize(12);
          doc.setFont(undefined, "bold");
          doc.setTextColor(50, 50, 50);
          doc.text(currentCategory, margin, y);
          y += 8;
        }

        // Field Label
        doc.setFontSize(10);
        doc.setFont(undefined, "bold");
        doc.setTextColor(0, 0, 0);
        doc.text(field.label, margin, y);
        y += 6;

        // Score Circle
        const circleX = margin + 5;
        doc.setDrawColor(0);
        const rgb =
          score >= state.highThreshold
            ? hexToRgb(state.colors.high)
```

```
            : score >= 4
            ? hexToRgb(state.colors.medium)
            : hexToRgb(state.colors.low);

        doc.setFillColor(...rgb);
        doc.circle(circleX, y + 5, 4, "FD");
        doc.setTextColor(255, 255, 255);
        doc.setFontSize(8);
        doc.text(String(score), circleX, y + 6, { align: "center" });

        y += 12;

        // Render matching text
        const renderTextBlock = (label, text, active) => {
          if (!text) return;
          doc.setFontSize(9);
          doc.setFont(undefined, "bold");
          doc.setTextColor(
            active ? 0 : 180,
            active ? 0 : 180,
            active ? 0 : 180
          );
          doc.text(`${label}:`, margin, y);
          y += 5;

          doc.setFont(undefined, "normal");
          const lines = doc.splitTextToSize(text, pageWidth - 2 * margin);
          lines.forEach((line) => {
            if (y + 6 > pageHeight - 15) {
              doc.addPage();
              y = 20;
            }
            doc.text(line, margin, y);
            y += 5;
          });
          y += 2;
        };

        renderTextBlock("HIGH", field.high, showHigh);
        renderTextBlock("NORMAL", field.normal, showNormal);
        renderTextBlock("LOW", field.low, showLow);

        y += 4;
      });

      // Footer (optional logos)
      const addLogos = () => {
        try {
          doc.addImage(leftLogoUrl, "PNG", 10, pageHeight - 20, 30, 10);
          doc.addImage(
            rightLogoUrl,
            "PNG",
            pageWidth - 40,
            pageHeight - 20,
            30,
            10
          );
        } catch (e) {
          console.warn("Logo failed to load. Skipping...");
        }
      };
      addLogos();

      // Save file
      doc.save("genomics-diet-report.pdf");
    },
    [state]
  );

  return { generatePDF };
};
```

■ File: src\hooks\useReportGeneration.js

```
================================================================================
```

```
  1: import { useState, useCallback } from 'react';
  2: import { useFormConfig } from '../contexts/FormConfigContext';
  3: import { isValidScore } from '../utils/helpers';
  4:
  5: /**
  6:  * Hook: useReportGeneration
  7:  * Transforms form input into structured report data based on score thresholds
  8:  */
  9: export const useReportGeneration = () => {
 10:   const { state } = useFormConfig(); // Get field config and settings from context
 11:   const [reportData, setReportData] = useState([]);
 12:
 13:   /**
 14:    * generateReport
 15:    * @param {Object} formData - { fieldId: score }
 16:    */
 17:   const generateReport = useCallback((formData) => {
 18:     const processedData = [];
 19:
 20:     // Loop through all fields from config
 21:     state.fields.forEach((field) => {
 22:       const rawValue = formData[field.id];
 23:       const score = parseInt(rawValue);
 24:
 25:       if (!isValidScore(score)) {
 26:         return; // Skip invalid scores
 27:       }
 28:
 29:       // Determine logic: high / normal / low
 30:       const isHigh = score >= state.highThreshold;
 31:       const isNormal = score >= 4 && score < state.highThreshold;
 32:       const isLow = score < 4;
 33:
 34:       processedData.push({
 35:         field,       // full field config
 36:         score,       // numeric score
 37:         showHigh: isHigh,
 38:         showNormal: isNormal,
 39:         showLow: isLow,
 40:       });
 41:     });
 42:
 43:     // Update state
 44:     setReportData(processedData);
 45:
 46:     // Return for immediate use
 47:     return processedData;
 48:   }, [state.fields, state.highThreshold]);
 49:
 50:   return { reportData, generateReport };
 51: };
```

```
--------------------------------------------------------------------------------
```

■ File: src\index.css

```
================================================================================
```

```
  1: /* Tailwind's base styles */
  2: @tailwind base;
  3: @tailwind components;
  4: @tailwind utilities;
  5:
  6: /* Custom global styles */
  7: @layer base {
  8:   /* HTML transition for smooth dark mode switching */
  9:   html {
 10:     transition: background-color 0.3s ease, color 0.3s ease;
 11:   }
 12:
 13:   /* Body settings for light and dark mode */
 14:   body {
 15:     @apply bg-white text-gray-900 dark:bg-gray-900 dark:text-white; /* Global body colors */
```

```
}

/* Customizations for links */
a {
  @apply text-blue-600 dark:text-blue-400; /* Links: light mode blue, dark mode lighter blue */
}

/* Buttons with default light/dark mode background */
button {
  @apply bg-gray-300 dark:bg-gray-700 text-gray-800 dark:text-gray-100; /* Default button styles */
}

/* Background color for any white-background elements */
.bg-white {
  @apply dark:bg-gray-900; /* Switches background color to dark mode */
}

/* Text color for general text */
.text-gray-900 {
  @apply dark:text-white; /* Changes text color to white in dark mode */
}

/* Form elements: inputs, textareas, and selects */
input,
textarea,
select {
  @apply bg-white dark:bg-gray-800 text-black dark:text-white border dark:border-gray-700;
}

/* Focused state of inputs, textareas, and selects */
input:focus,
textarea:focus,
select:focus {
  @apply ring-2 ring-blue-500 dark:ring-blue-300;
}

/* Global font and smoothing */
body {
  margin: 0;
  font-family: -apple-system, BlinkMacSystemFont, "Segoe UI", "Roboto",
    "Oxygen", "Ubuntu", "Cantarell", "Fira Sans", "Droid Sans",
    "Helvetica Neue", sans-serif;
  -webkit-font-smoothing: antialiased;
  -moz-osx-font-smoothing: grayscale;
}

/* Code styles for code blocks */
code {
  font-family: source-code-pro, Menlo, Monaco, Consolas, "Courier New",
    monospace;
}

/* Global styling for text input elements */
input[type="text"],
input[type="number"],
input[type="email"],
textarea,
select {
  @apply border-2 rounded-lg p-2 dark:border-gray-600;
}

/* Button hover and focus states */
button:hover {
  @apply bg-gray-200 dark:bg-gray-700; /* Darker button on hover */
}

/* Ensuring links are visible in dark mode */
.text-blue-600 {
  @apply dark:text-blue-400; /* Light blue in normal mode, changes to darker in dark mode */
}

/* Customize card elements for dark mode */
.card {
```

```
 89:     @apply bg-white dark:bg-gray-800 text-black dark:text-white border dark:border-gray-700;
 90:   }
 91:
 92:   /* Customize borders */
 93:   .border-gray-300 {
 94:     @apply dark:border-gray-600; /* Change border color in dark mode */
 95:   }
 96:
 97:   /* Ensure smooth transitions when toggling dark/light mode */
 98:   .transition-all {
 99:     transition: all 0.3s ease; /* Apply smooth transitions */
100:   }
101: }
```

■ File: src\index.js

```
  1: import React from 'react';
  2: import ReactDOM from 'react-dom/client';
  3: import './index.css';
  4: import App from './App';
  5: import reportWebVitals from './reportWebVitals';
  6:
  7: const root = ReactDOM.createRoot(document.getElementById('root'));
  8: root.render(
  9:   <React.StrictMode>
 10:     <App />
 11:   </React.StrictMode>
 12: );
 13:
 14: // If you want to start measuring performance in your app, pass a function
 15: // to log results (for example: reportWebVitals(console.log))
 16: // or send to an analytics endpoint. Learn more: https://bit.ly/CRA-vitals
 17: reportWebVitals();
```

■ File: src\logo.svg

```
<svg xmlns="http://www.w3.org/2000/svg" viewBox="0 0 841.9 595.3"><g fill="#61DAFB"><path d="M666.3 296.5c0-32.5
```

■ File: src\reportWebVitals.js

```
  1: const reportWebVitals = onPerfEntry => {
  2:   if (onPerfEntry && onPerfEntry instanceof Function) {
  3:     import('web-vitals').then(({ getCLS, getFID, getFCP, getLCP, getTTFB }) => {
  4:       getCLS(onPerfEntry);
  5:       getFID(onPerfEntry);
  6:       getFCP(onPerfEntry);
  7:       getLCP(onPerfEntry);
  8:       getTTFB(onPerfEntry);
  9:     });
 10:   }
 11: };
 12:
 13: export default reportWebVitals;
```

■ File: src\setupTests.js

```
  1: // jest-dom adds custom jest matchers for asserting on DOM nodes.
  2: // allows you to do things like:
  3: // expect(element).toHaveTextContent(/react/i)
  4: // learn more: https://github.com/testing-library/jest-dom
  5: import '@testing-library/jest-dom';
```

■ File: src\utils\constants.js

```
================================================================================

--------------------------------------------------------------------------------
```

■ File: src\utils\helpers.js

```
================================================================================
  1: // Convert HEX color to RGB array for jsPDF
  2: export function hexToRgb(hex) {
  3:   const result = /^#?([a-f\d]{2})([a-f\d]{2})([a-f\d]{2})$/i.exec(hex);
  4:   return result
  5:     ? [
  6:         parseInt(result[1], 16),
  7:         parseInt(result[2], 16),
  8:         parseInt(result[3], 16)
  9:       ]
 10:     : [0, 0, 0];
 11: }
 12:
 13: // Simple validation helper
 14: export const isValidScore = (value) => {
 15:   const num = parseInt(value);
 16:   return !isNaN(num) && num >= 1 && num <= 10;
 17: };

--------------------------------------------------------------------------------
```

■ File: tailwind.config.js

```
================================================================================
  1: /** @type {import('tailwindcss').Config} */
  2: module.exports = {
  3:   darkMode: "class", // ? enables class-based dark mode
  4:   content: ["./src/**/*.{js,jsx}"],
  5:   theme: {
  6:     extend: {},
  7:   },
  8:   plugins: [],
  9: };

--------------------------------------------------------------------------------
```