■ Project Code Export

■ Frontend File List:

```
- .gitignore
- 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\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\SettingsPanel.js
- src\contexts\FormConfigContext.js
- src\data\formConfig.json
- src\hooks\useConfigImportExport.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\styles\index.css
- src\utils\constants.js
- src\utils\helpers.js
- tailwind.config.js
```

■ File: .gitignore

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

■ File: postcss.config.js

```js
module.exports = {
  plugins: {
    tailwindcss: {},
    autoprefixer: {},
  },
}
```

■ File: public\favicon.ico

```
[Binary file - .ico format]
```

■ File: public\index.html

```html
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="utf-8" />
    <link rel="icon" href="%PUBLIC_URL%/favicon.ico" />
    <meta name="viewport" content="width=device-width, initial-scale=1" />
    <meta name="theme-color" content="#000000" />
    <meta
      name="description"
      content="Web site created using create-react-app"
    />
    <link rel="apple-touch-icon" href="%PUBLIC_URL%/logo192.png" />
    <!--
      manifest.json provides metadata used when your web app is installed on a
      user's mobile device or desktop. See https://developers.google.com/web/fundamentals/web-app-manifest/
    -->
    <link rel="manifest" href="%PUBLIC_URL%/manifest.json" />
    <!--
      Notice the use of %PUBLIC_URL% in the tags above.
      It will be replaced with the URL of the `public` folder during the build.
      Only files inside the `public` folder can be referenced from the HTML.

      Unlike "/favicon.ico" or "favicon.ico", "%PUBLIC_URL%/favicon.ico" will
      work correctly both with client-side routing and a non-root public URL.
      Learn how to configure a non-root public URL by running `npm run build`.
    -->
    <title>React App</title>
  </head>
  <body>
    <noscript>You need to enable JavaScript to run this app.</noscript>
    <div id="root"></div>
    <!--
      This HTML file is a template.
      If you open it directly in the browser, you will see an empty page.

      You can add webfonts, meta tags, or analytics to this file.
      The build step will place the bundled scripts into the <body> tag.

      To begin the development, run `npm start` or `yarn start`.
      To create a production bundle, use `npm run build` or `yarn build`.
    -->
  </body>
</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

```
================================================================================
{
  "short_name": "React App",
  "name": "Create React App Sample",
  "icons": [
    {
      "src": "favicon.ico",
      "sizes": "64x64 32x32 24x24 16x16",
      "type": "image/x-icon"
    },
    {
      "src": "logo192.png",
      "type": "image/png",
      "sizes": "192x192"
    },
    {
      "src": "logo512.png",
      "type": "image/png",
      "sizes": "512x512"
    }
  ],
  "start_url": ".",
  "display": "standalone",
  "theme_color": "#000000",
  "background_color": "#ffffff"
}

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

■ File: public\right.png

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

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

■ File: public\robots.txt

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

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

■ File: script.py

```
================================================================================
import os
from reportlab.lib.pagesizes import A4
from reportlab.lib.units import mm
from reportlab.pdfgen import canvas

def get_code_files(directory, excluded_files=None, excluded_dirs=None):
    """Fetch all project files except specified exclusions."""
    if excluded_files is None:
        excluded_files = {'package.json', 'package-lock.json'}

    if excluded_dirs is None:
        excluded_dirs = {'node_modules', '.git', '__pycache__', 'build', '.next', 'dist'}

    code_files = {}

    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',
        '*.log'
    }

    print("? Scanning project files...")
    code_files = get_code_files(root_dir, excluded_files, excluded_dirs)

    if not code_files:
        print("? No files found to process!")
        return

    print(f"? Found {len(code_files)} files to include in PDF")
    create_pdf(code_files)


if __name__ == "__main__":
    main()
```

■ File: src\App.css

==

```
.App {
  text-align: center;
}

.App-logo {
  height: 40vmin;
  pointer-events: none;
}

@media (prefers-reduced-motion: no-preference) {
  .App-logo {
    animation: App-logo-spin infinite 20s linear;
  }
}

.App-header {
  background-color: #282c34;
  min-height: 100vh;
  display: flex;
  flex-direction: column;
  align-items: center;
  justify-content: center;
  font-size: calc(10px + 2vmin);
  color: white;
}

.App-link {
  color: #61dafb;
}

@keyframes App-logo-spin {
  from {
    transform: rotate(0deg);
  }
  to {
    transform: rotate(360deg);
  }
}
```

■ File: src\App.js

==

```
import React from 'react';
import { FormConfigProvider } from './contexts/FormConfigContext';
import AppContent from './AppContent';
import Toasts from './components/common/Toasts';

function App() {
  return (
    <FormConfigProvider>
      <AppContent />
      <Toasts />
    </FormConfigProvider>
  );
}

export default App;
```

■ File: src\App.test.js

==

```
import { render, screen } from '@testing-library/react';
import App from './App';

test('renders learn react link', () => {
  render(<App />);
  const linkElement = screen.getByText(/learn react/i);
  expect(linkElement).toBeInTheDocument();
});
```

■ File: src\AppContent.js

==

```
import React, { useState } from 'react';
import TabNavigation from './components/common/TabNavigation';
import InputForm from './components/form/InputForm';
import ReportOutput from './components/report/ReportOutput';
import AdminPanel from './components/admin/AdminPanel';
import SettingsPanel from './components/settings/SettingsPanel';
import { useReportGeneration } from './hooks/useReportGeneration';
import { AnimatePresence, motion } from 'framer-motion';

const AppContent = () => {
  const [activeTab, setActiveTab] = useState('form');
  const { reportData, generateReport } = useReportGeneration();

  const handleGenerateReport = (formData) => {
    generateReport(formData);
  };

  const tabTransition = {
    initial: { opacity: 0, y: 20 },
    animate: { opacity: 1, y: 0 },
    exit: { opacity: 0, y: -20 },
    transition: { duration: 0.3 }
  };

  return (
    <div className="bg-gray-100 font-sans min-h-screen">
      <TabNavigation activeTab={activeTab} setActiveTab={setActiveTab} />

      <AnimatePresence mode="wait">
        {activeTab === 'form' && (
          <motion.div key="form" {...tabTransition}>
            <div className="flex flex-col md:flex-row h-screen desktop-layout">
              <InputForm onGenerateReport={handleGenerateReport} reportData={reportData} />
              <ReportOutput reportData={reportData} />
            </div>
          </motion.div>
        )}

        {activeTab === 'admin' && (
          <motion.div key="admin" {...tabTransition}>
            <div className="p-6">
              <AdminPanel />
            </div>
          </motion.div>
        )}

        {activeTab === 'settings' && (
          <motion.div key="settings" {...tabTransition}>
            <div className="p-6">
              <SettingsPanel />
            </div>
          </motion.div>
        )}
      </AnimatePresence>
    </div>
  );
};

export default AppContent;
```

■ File: src\assets\placeholder.svg

==

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

==

```
import React from "react";
import { useFormConfig } from "../../contexts/FormConfigContext";

const AdminPanel = () => {
  const { state, dispatch } = useFormConfig();

  // ? Add a new field template
  const addNewField = () => {
    const newField = {
      id: `field_${Date.now()}`,
      label: "New Field",
      category: "",
      high: "High score recommendation",
      normal: "Normal score recommendation",
      low: "Low score recommendation",
    };

    dispatch({ type: "ADD_FIELD", field: newField });
  };

  // ?? Delete field by index
  const deleteField = (index) => {
    if (window.confirm("Are you sure you want to delete this field?")) {
      dispatch({ type: "DELETE_FIELD", index });
    }
  };

  // ?? Update specific property in a field
  const updateField = (index, property, value) => {
    dispatch({ type: "UPDATE_FIELD", index, property, value });
  };

  return (
    <div className="max-w-6xl mx-auto px-4 py-8">
      <div className="flex justify-between items-center mb-6">
        <h2 className="text-2xl font-bold">?? Field Management</h2>
        <button
          onClick={addNewField}
          className="bg-green-600 hover:bg-green-700 text-white px-5 py-2 rounded-lg font-semibold"
        >
          ? Add New Field
        </button>
      </div>

      {/* List all fields */}
      <div className="space-y-6">
        {state.fields.map((field, index) => (
          <div
            key={field.id}
            className="border rounded-lg p-6 bg-white shadow-sm space-y-4"
          >
            <div className="flex justify-between items-center">
              <h3 className="text-lg font-semibold">
                Field {index + 1}: {field.label}
              </h3>
              <button
                onClick={() => deleteField(index)}
                className="text-red-600 hover:text-red-800 font-medium"
              >
                ?? Delete
              </button>
            </div>

            <div className="grid grid-cols-1 md:grid-cols-2 gap-4">
              {/* ID */}
              <div>
                <label className="block text-sm font-medium mb-1">
                  Field ID
                </label>
                <input
                  type="text"
                  value={field.id}
                  onChange={(e) => updateField(index, "id", e.target.value)}
                  className="w-full border border-gray-300 rounded px-3 py-2"
                />
              </div>

              {/* Label */}
              <div>
                <label className="block text-sm font-medium mb-1">Label</label>
                <input
                  type="text"
                  value={field.label}
                  onChange={(e) => updateField(index, "label", e.target.value)}
                  className="w-full border border-gray-300 rounded px-3 py-2"
                />
              </div>

              {/* Category */}
              <div>
                <label className="block text-sm font-medium mb-1">
                  Category
                </label>
                <input
                  type="text"
                  value={field.category}
                  onChange={(e) =>
                    updateField(index, "category", e.target.value)
                  }
                  className="w-full border border-gray-300 rounded px-3 py-2"
                />
              </div>

              {/* HIGH Recommendation */}
              <div>
                <label className="block text-sm font-medium mb-1">
                  High Recommendation
                </label>
                <textarea
                  value={field.high}
                  onChange={(e) => updateField(index, "high", e.target.value)}
                  className="w-full border border-gray-300 rounded px-3 py-2"
                  rows={3}
                />
              </div>

              {/* NORMAL Recommendation */}
              <div>
                <label className="block text-sm font-medium mb-1">
                  Normal Recommendation
                </label>
                <textarea
                  value={field.normal}
                  onChange={(e) => updateField(index, "normal", e.target.value)}
                  className="w-full border border-gray-300 rounded px-3 py-2"
                  rows={3}
                />
              </div>

              {/* LOW Recommendation */}
              <div>
                <label className="block text-sm font-medium mb-1">
                  Low Recommendation
                </label>
                <textarea
                  value={field.low}
                  onChange={(e) => updateField(index, "low", e.target.value)}
                  className="w-full border border-gray-300 rounded px-3 py-2"
                  rows={3}
                />
              </div>
            </div>
          </div>
        ))}
      </div>
    </div>
  );
};

export default AdminPanel;
```

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

==

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

==

```
import React from "react";

const TabNavigation = ({ activeTab, setActiveTab }) => {
  const tabs = [
    { id: "form", label: "? Form View" },
    { id: "admin", label: "?? Admin Panel" },
    { id: "settings", label: "?? Form Settings" },
  ];

  return (
    <div className="bg-white shadow-sm border-b sticky top-0 z-10">
      <div className="container mx-auto px-4">
        <div className="flex gap-2 py-4">
          {tabs.map((tab) => (
            <button
              key={tab.id}
              className={`px-4 py-2 rounded-lg transition-all font-medium ${
                activeTab === tab.id
                  ? "bg-green-600 text-white"
                  : "bg-gray-200 text-gray-700 hover:bg-gray-300"
              }`}
              onClick={() => setActiveTab(tab.id)}
            >
              {tab.label}
            </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-right"
      autoClose={3000}
      hideProgressBar={false}
      newestOnTop={false}
      closeOnClick
      pauseOnFocusLoss
      draggable
      pauseOnHover
      theme="colored"
    />
  );
};

export default Toasts;
```

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

==

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

==

```
import React, { useState, useEffect } from "react";
import { useFormConfig } from "../../contexts/FormConfigContext";
import { usePDFGeneration } from "../../hooks/usePDFGeneration";
import { isValidScore } from "../../utils/helpers";
import { toast } from "react-toastify";

const LOCAL_STORAGE_KEY = "genomics_form_data";

const InputForm = ({ onGenerateReport, reportData }) => {
  const { state } = useFormConfig();
  const { generatePDF } = usePDFGeneration();

  // Try restoring from localStorage
  const [formData, setFormData] = useState(() => {
    try {
      const stored = localStorage.getItem(LOCAL_STORAGE_KEY);
      return stored ? JSON.parse(stored) : {};
    } catch (e) {
      return {};
    }
  });

  const [errors, setErrors] = useState({});

  // ? Save to localStorage on formData change
  useEffect(() => {
    localStorage.setItem(LOCAL_STORAGE_KEY, JSON.stringify(formData));
  }, [formData]);

  const handleInputChange = (fieldId, value) => {
    const updatedValue = value.replace(/\D/g, "");
    setFormData((prev) => ({
      ...prev,
      [fieldId]: updatedValue,
    }));
    setErrors((prev) => ({
      ...prev,
      [fieldId]: null,
    }));
  };

  const validateForm = () => {
    const newErrors = {};
    state.fields.forEach((field) => {
      const value = formData[field.id];
      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 handleDownloadPDF = () => {
    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>

          <button
            type="button"
            onClick={handleDownloadPDF}
            className="bg-blue-600 hover:bg-blue-700 text-white px-6 py-3 rounded-lg font-semibold"
          >
            Download PDF
          </button>

          <button
            type="button"
            onClick={handleClearForm}
            className="bg-gray-500 hover:bg-gray-600 text-white px-6 py-3 rounded-lg font-semibold"
          >
            Clear Input
          </button>
        </div>
      </form>
    </div>
  );
};

export default InputForm;
```

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

==

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

==

```
import React from "react";
import { useFormConfig } from "../../contexts/FormConfigContext";

const ReportOutput = ({ reportData }) => {
  const { state } = useFormConfig();

  // Assign color to score
  const getScoreColor = (score) => {
    if (score >= state.highThreshold) return "bg-red-600";
    if (score >= 4) return "bg-yellow-500";
    return "bg-green-600";
  };

  // Style active/inactive text
  const getTextStyle = (isActive) =>
    isActive ? "text-gray-900 font-semibold" : "text-gray-400";

  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">
        {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-xs 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
                      )}`}
                    >
                      LOW
                    </div>
                    <div
                      className={`text-xs leading-tight ${getTextStyle(showLow)}`}
                    >
                      {field.low.split("\n").map((line, i, arr) => (
                        <React.Fragment key={i}>
                          {line}
                          {i < arr.length - 1 && <br />}
                        </React.Fragment>
                      ))}
                    </div>
                  </div>
                </div>
              </div>
            );

            return elements;
          })}
        </div>
      </div>
  );
};

export default ReportOutput;
```

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

==

```
import React, { useState } from "react";
import { useFormConfig } from "../../contexts/FormConfigContext";

const SettingsPanel = () => {
  const { state, dispatch } = useFormConfig();

  // Internal editable state
  const [settings, setSettings] = useState({
    title: state.title,
    quote: state.quote,
    description: state.description,
    headerColor: state.headerColor,
    highThreshold: state.highThreshold,
    colors: {
      low: state.colors.low,
      medium: state.colors.medium,
      high: state.colors.high,
    },
  });

  // Change handlers
  const handleChange = (key, value) => {
    setSettings((prev) => ({
      ...prev,
      [key]: value,
    }));
  };

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

  // Apply changes to context
  const applySettings = () => {
    dispatch({ type: "UPDATE_SETTINGS", settings });
    alert("? Settings applied successfully!");
  };

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

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

      <div className="grid grid-cols-1 md:grid-cols-2 gap-6">
        {/* Left Section */}
        <div className="bg-white rounded-lg shadow-sm p-6 space-y-4">
          <h3 className="text-lg font-semibold">? Header Info</h3>

          <div>
            <label className="block text-sm font-medium mb-1">Main Title</label>
            <input
              type="text"
              value={settings.title}
              onChange={(e) => handleChange("title", e.target.value)}
              className="w-full border rounded px-3 py-2"
            />
          </div>

          <div>
            <label className="block text-sm font-medium mb-1">Quote</label>
            <textarea
              rows={2}
              value={settings.quote}
              onChange={(e) => handleChange("quote", e.target.value)}
              className="w-full border rounded px-3 py-2"
            />
          </div>

          <div>
            <label className="block text-sm font-medium mb-1">
              Description
            </label>
            <textarea
              rows={4}
              value={settings.description}
              onChange={(e) => handleChange("description", e.target.value)}
              className="w-full border rounded px-3 py-2"
            />
          </div>

          <div>
```

```
            <label className="block text-sm font-medium mb-1">
              Header Background Color
            </label>
            <input
              type="color"
              value={settings.headerColor}
              onChange={(e) => handleChange("headerColor", e.target.value)}
              className="h-10 w-full border rounded"
            />
          </div>
        </div>

        {/* Right Section */}
        <div className="bg-white rounded-lg shadow-sm p-6 space-y-4">
          <h3 className="text-lg font-semibold">? Score Logic</h3>

          <div>
            <label className="block text-sm font-medium mb-1">
              High Score Threshold (?)
            </label>
            <input
              type="number"
              min="1"
              max="10"
              value={settings.highThreshold}
              onChange={(e) =>
                handleChange("highThreshold", parseInt(e.target.value))
              }
              className="w-full border rounded px-3 py-2"
            />
          </div>

          <h3 className="text-lg font-semibold mt-6">? Score Colors</h3>

          <div className="space-y-2">
            <div>
              <label className="block text-sm font-medium mb-1">High</label>
              <input
                type="color"
                value={settings.colors.high}
                onChange={(e) => handleColorChange("high", e.target.value)}
                className="h-10 w-full border rounded"
              />
            </div>

            <div>
              <label className="block text-sm font-medium mb-1">Medium</label>
              <input
                type="color"
                value={settings.colors.medium}
                onChange={(e) => handleColorChange("medium", e.target.value)}
                className="h-10 w-full border rounded"
              />
            </div>

            <div>
              <label className="block text-sm font-medium mb-1">Low</label>
              <input
                type="color"
                value={settings.colors.low}
                onChange={(e) => handleColorChange("low", e.target.value)}
                className="h-10 w-full border rounded"
              />
            </div>
          </div>
        </div>
      </div>

      {/* Buttons */}
      <div className="mt-6 flex 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>
      </div>
    </div>
  );
};

export default SettingsPanel;
```

■ File: src\contexts\FormConfigContext.js

==

```
import React, { createContext, useContext, useReducer } from "react";
import initialConfig from "../data/formConfig.json";

// --- Reducer Function ---
const formConfigReducer = (state, action) => {
  switch (action.type) {
    case "UPDATE_FIELD":
      return {
        ...state,
        fields: state.fields.map((field, i) =>
          i === action.index
            ? { ...field, [action.property]: action.value }
            : field
        ),
      };

    case "ADD_FIELD":
      return {
        ...state,
        fields: [...state.fields, action.field],
      };

    case "DELETE_FIELD":
      return {
        ...state,
        fields: state.fields.filter((_, i) => i !== action.index),
      };

    case "UPDATE_SETTINGS":
      return {
        ...state,
        ...action.settings,
      };

    case "IMPORT_CONFIG":
      return {
        ...state,
        ...action.config,
      };

    default:
      return state;
  }
};

// --- Context Creation ---
const FormConfigContext = createContext(null);

// --- Provider ---
export const FormConfigProvider = ({ children }) => {
  const [state, dispatch] = useReducer(formConfigReducer, initialConfig);

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

// --- Custom Hook ---
export const useFormConfig = () => {
  const context = useContext(FormConfigContext);
  if (!context) {
    throw new Error("useFormConfig must be used within a FormConfigProvider");
  }
  return context;
};
```

■ File: src\data\formConfig.json

==

```
{
  "title": "GENOMICS & DIET",
  "quote": "\"YOU ARE WHAT YOU EAT\" - Victor Lindlahr",
  "description": "A right diet is the one that makes you feel happy, keeps you healthy, does not make you f
  "headerColor": "#16a34a",
  "colors": {
    "low": "#16a34a",
    "medium": "#f59e0b",
    "high": "#dc2626"
  },
  "highThreshold": 6,
  "categories": ["MACRONUTRIENTS", "MEAL PATTERN", "FOOD SENSITIVITIES"],
  "fields": [
    {
      "id": "carb",
      "label": "Carbohydrate Sensitivity",
      "category": "MACRONUTRIENTS",
      "high": "Maintain carb intake <45%\nFor obesity & IR control",
      "normal": "Maintain carb intake <50%\nBalanced recommendation",
      "low": "Maintain carb intake <60%\nFor obesity & IR control"
    },
    {
      "id": "fat",
      "label": "Fat Sensitivity",
      "category": "MACRONUTRIENTS",
      "high": "Fat intake not to exceed\n15% of total calories",
      "normal": "Fat intake should be\n20% of total calories",
      "low": "Fat intake advised up to\n25% of total calories"
    },
    {
      "id": "protein",
      "label": "Protein Requirement",
      "category": "MACRONUTRIENTS",
      "high": "Protein supplements\nneeded along with dietary\nsource",
      "normal": "Maintain adequate protein\nthrough balanced diet and\noccasional supplements",
      "low": "Protein supplements not\nneeded, intake through\ndiet is enough"
    }
  ]
}
```

■ File: src\hooks\useConfigImportExport.js

==

```
import { useCallback } from "react";
import { useFormConfig } from "../contexts/FormConfigContext";

export const useConfigImportExport = () => {
  const { state, dispatch } = useFormConfig();

  const exportConfig = useCallback(() => {
    const dataStr = JSON.stringify(state, null, 2);
    const blob = new Blob([dataStr], { type: "application/json" });
    const url = URL.createObjectURL(blob);

    const link = document.createElement("a");
    link.href = url;
    link.download = "genomics-form-config.json";
    link.click();
    URL.revokeObjectURL(url);
  }, [state]);

  const importConfig = useCallback(
    (file) => {
      const reader = new FileReader();
      reader.onload = (e) => {
        try {
          const parsed = JSON.parse(e.target.result);
          dispatch({ type: "IMPORT_CONFIG", config: parsed });
          alert("Configuration imported successfully.");
        } catch (err) {
          alert("Error importing config: " + err.message);
        }
      };
      reader.readAsText(file);
    },
    [dispatch]
  );

  return { exportConfig, importConfig };
};
```

■ File: src\hooks\usePDFGeneration.js

==

```
import { useCallback } from "react";
import { jsPDF } from "jspdf";
import { useFormConfig } from "../contexts/FormConfigContext";
import { hexToRgb } from "../utils/helpers";

export const usePDFGeneration = () => {
  const { state } = useFormConfig();

  // Base64 logo (optional)
  const leftLogoUrl = "/left.png";
  const rightLogoUrl = "/right.png";

  const generatePDF = useCallback(
    (reportData) => {
      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

==

```
import { useState, useCallback } from 'react';
import { useFormConfig } from '../contexts/FormConfigContext';
import { isValidScore } from '../utils/helpers';

/**
 * Hook: useReportGeneration
 * Transforms form input into structured report data based on score thresholds
 */
export const useReportGeneration = () => {
  const { state } = useFormConfig(); // Get field config and settings from context
  const [reportData, setReportData] = useState([]);

  /**
   * generateReport
   * @param {Object} formData - { fieldId: score }
   */
  const generateReport = useCallback((formData) => {
    const processedData = [];

    // Loop through all fields from config
    state.fields.forEach((field) => {
      const rawValue = formData[field.id];
      const score = parseInt(rawValue);

      if (!isValidScore(score)) {
        return; // Skip invalid scores
      }

      // Determine logic: high / normal / low
      const isHigh = score >= state.highThreshold;
      const isNormal = score >= 4 && score < state.highThreshold;
      const isLow = score < 4;

      processedData.push({
        field,       // full field config
        score,       // numeric score
        showHigh: isHigh,
        showNormal: isNormal,
        showLow: isLow,
      });
    });

    // Update state
    setReportData(processedData);

    // Return for immediate use
    return processedData;
  }, [state.fields, state.highThreshold]);

  return { reportData, generateReport };
};
```

■ File: src\index.css

==

```css
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 {
  font-family: source-code-pro, Menlo, Monaco, Consolas, "Courier New",
    monospace;
}
@tailwind base;
@tailwind components;
@tailwind utilities;
```

■ File: src\index.js

==

```js
import React from 'react';
import ReactDOM from 'react-dom/client';
import './index.css';
import App from './App';
import reportWebVitals from './reportWebVitals';

const root = ReactDOM.createRoot(document.getElementById('root'));
root.render(
  <React.StrictMode>
    <App />
  </React.StrictMode>
);

// If you want to start measuring performance in your app, pass a function
// to log results (for example: reportWebVitals(console.log))
// or send to an analytics endpoint. Learn more: https://bit.ly/CRA-vitals
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

```
================================================================================
const reportWebVitals = onPerfEntry => {
  if (onPerfEntry && onPerfEntry instanceof Function) {
    import('web-vitals').then(({ getCLS, getFID, getFCP, getLCP, getTTFB }) => {
      getCLS(onPerfEntry);
      getFID(onPerfEntry);
      getFCP(onPerfEntry);
      getLCP(onPerfEntry);
      getTTFB(onPerfEntry);
    });
  }
};

export default reportWebVitals;

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

■ File: src\setupTests.js

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

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

■ File: src\styles\index.css

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

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

■ File: src\utils\constants.js

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

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

■ File: src\utils\helpers.js

```
================================================================================
// Convert HEX color to RGB array for jsPDF
export function hexToRgb(hex) {
  const result = /^#?([a-f\d]{2})([a-f\d]{2})([a-f\d]{2})$/i.exec(hex);
  return result
    ? [
        parseInt(result[1], 16),
        parseInt(result[2], 16),
        parseInt(result[3], 16)
      ]
    : [0, 0, 0];
}

// Simple validation helper
export const isValidScore = (value) => {
  const num = parseInt(value);
  return !isNaN(num) && num >= 1 && num <= 10;
};

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

■ File: tailwind.config.js

```
================================================================================
/** @type {import('tailwindcss').Config} */
module.exports = {
  content: [
    "./src/**/*.{js,jsx,ts,tsx}",
  ],
  theme: {
    extend: {},
  },
  plugins: [],
}
```
