TREDENCE PROCESS - FULL STACK DEVELOPER INTERN ASSIGNMENT

Title: HR Workflow Designer - Visual HR Process Builder

1. **PROJECT OVERVIEW:**

This project is a visual HR Workflow Designer application that allows users to create and configure HR processes such as onboarding, approval cycles, automated tasks, and completion steps.
The tool provides a drag-and-drop interface where users can create five types of workflow nodes:

- Start Node – defines the beginning of the process
- Task Node – represents manual work assigned to an employee
- Approval Node – captures a manager/HR approval step
- Automated Node – performs system-driven automated tasks
- End Node – marks the completion of the workflow

Users can connect these nodes to build a complete HR workflow, edit node details, simulate process execution, export workflow data as JSON, and import previously created workflows.

This tool demonstrates how HR workflows can be digitized and simplified without writing code.

2. **FEATURES IMPLEMENTED:**

Below are the key features included in this application:

a. Add Workflow Nodes

Users can add Start, Task, Approval, Automated, and End nodes using the left-side palette.

b. Drag & Drop Node Positioning

Nodes can be freely moved across the canvas.

c. Connect Nodes

Users can connect nodes by dragging from one connection point (handle) to another to create a directional workflow.

d. Node Configuration Panel

Clicking any node opens its editable configuration on the right side.
Users can change:

- Label
- Title

- Description
- Assignee
- Approver role
- Automated action ID
- End message
- Custom fields

e. Workflow Simulation

Displays a step-by-step textual simulation of the HR process.

f. Workflow Export to JSON

Generates a structured JSON file containing:

- Node details
- Node positions
- Node configuration
- Workflow edges

g. Workflow Import

Allows loading previously saved workflows.

h. Delete Node Feature

Pressing Delete or Backspace removes the selected node and its connected edges.

i. Clean UI & Interactive Controls

Mini - map, zoom controls, and grid background enhance usability.

3. Simple Steps on How to Execute This Assignment

Step 1: Open Command Prompt or cmd

Step 2: Just Paste this Command in cmd - **cd C:\**

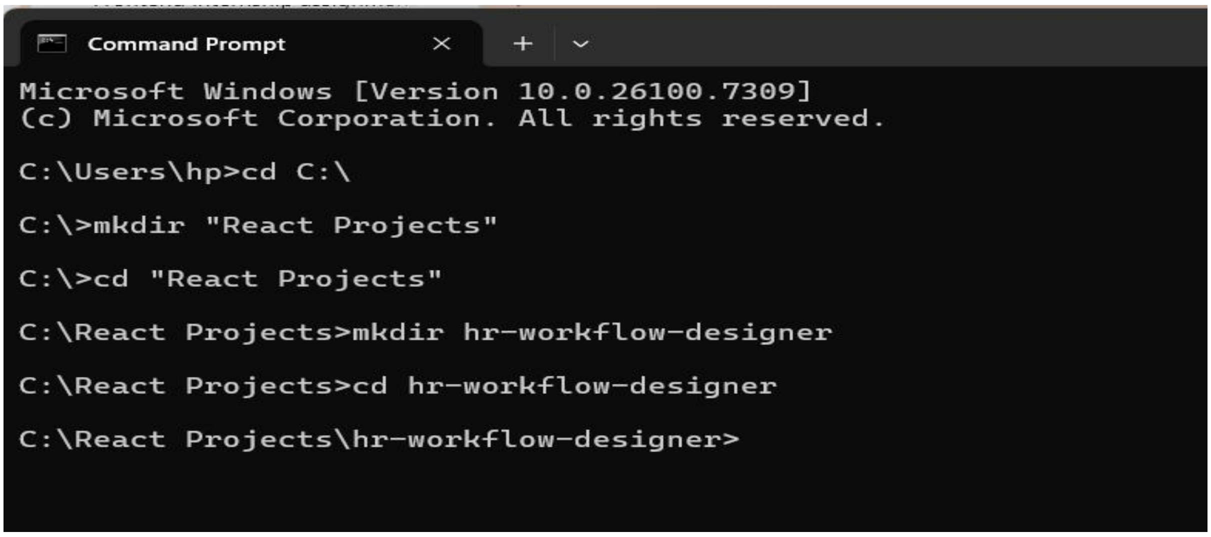
Step 3: Paste this Command in cmd - **mkdir "React Projects"**

Step 4: **cd "React Projects"** (Paste this again in cmd)

Step 5: C:\React Projects>

Step 6: mkdir hr-workflow-designer

Step 7: cd hr-workflow-designer

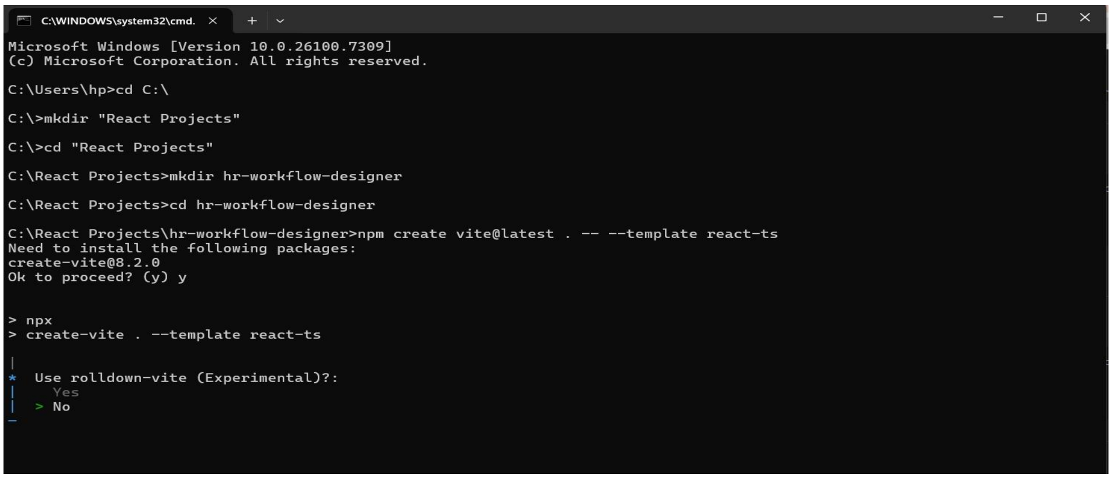


```
Microsoft Windows [Version 10.0.26100.7309]
(c) Microsoft Corporation. All rights reserved.

C:\Users\hp>cd C:\

C:\>mkdir "React Projects"

C:\>cd "React Projects"

C:\React Projects>mkdir hr-workflow-designer

C:\React Projects>cd hr-workflow-designer

C:\React Projects\hr-workflow-designer>
```

Step 8: Now run the React project setup: Same just type this Command in cmd : **npm create vite@latest . -- --template react-ts**

Step 9: Then it will ask - Ok to proceed? (y) : Press y and Enter

Step 10:

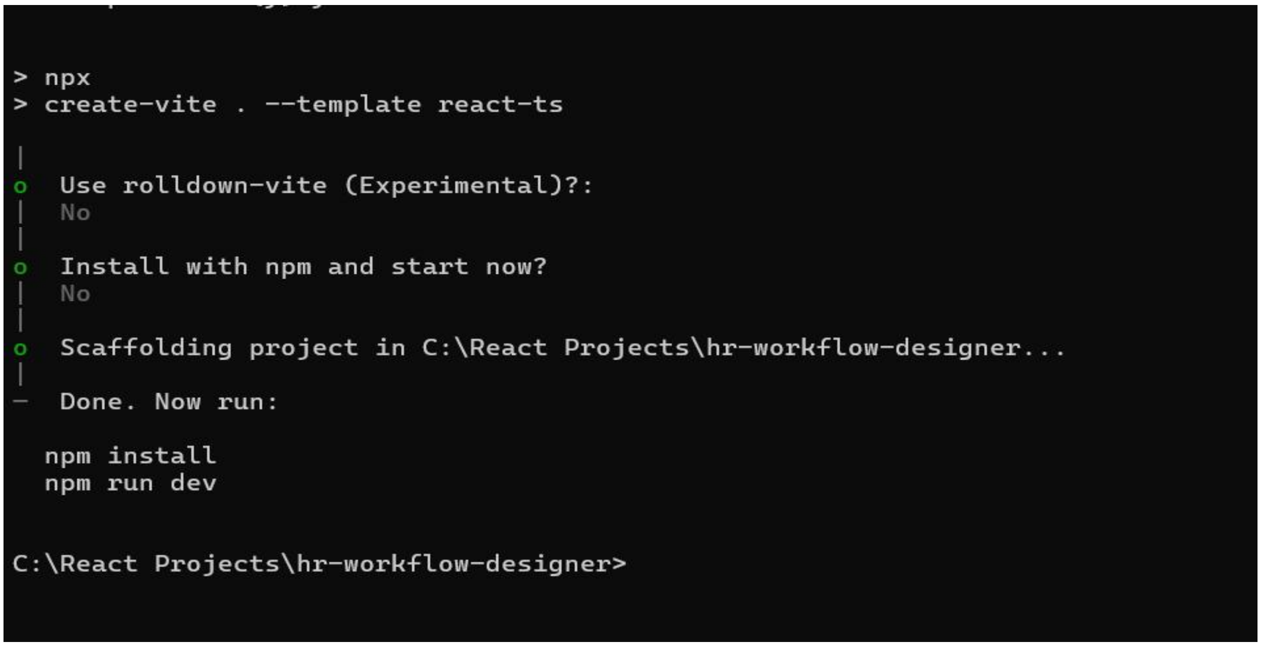


```
Microsoft Windows [Version 10.0.26100.7309]
(c) Microsoft Corporation. All rights reserved.

C:\Users\hp>cd C:\

C:\>mkdir "React Projects"

C:\>cd "React Projects"

C:\React Projects>mkdir hr-workflow-designer

C:\React Projects>cd hr-workflow-designer

C:\React Projects\hr-workflow-designer>npm create vite@latest . -- --template react-ts
Need to install the following packages:
create-vite@8.2.0
Ok to proceed? (y) y


> npx
> create-vite . --template react-ts

|
*  Use rolldown-vite (Experimental)?:
|    Yes
|  > No
—
```

If u see in above image Yes or No is there Just Type No there**.**

Step 11: Again, Select No

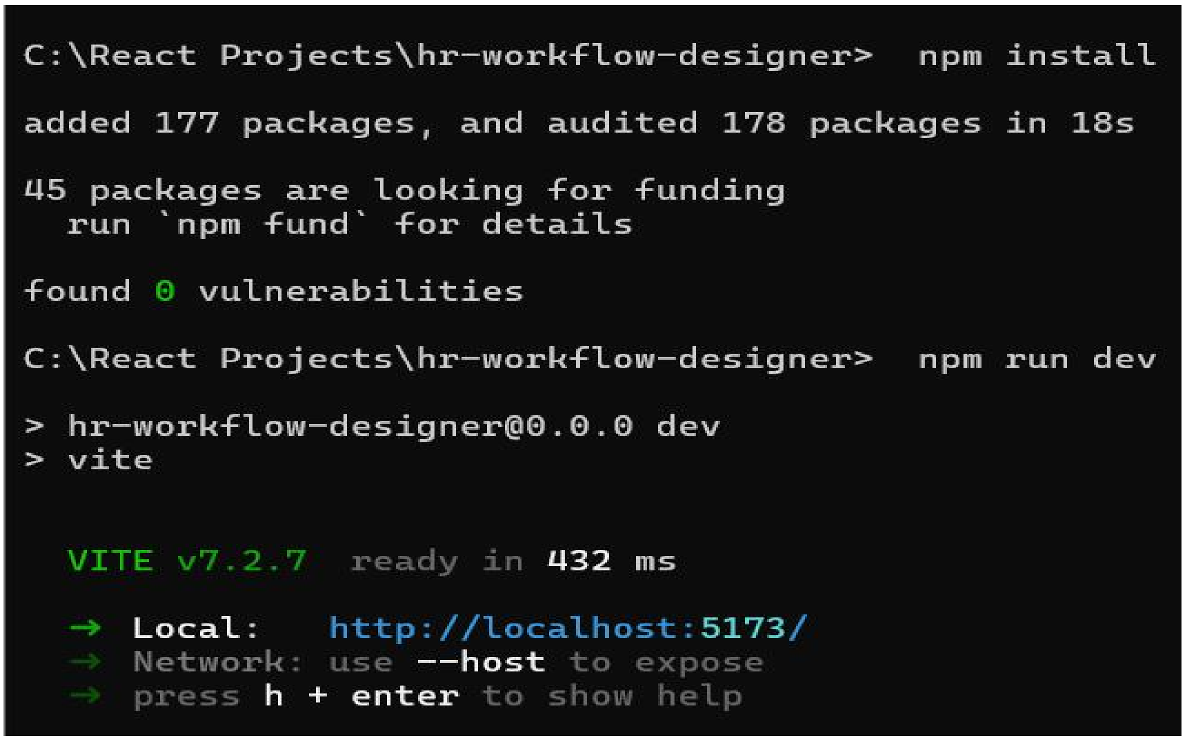
- **Arrow key ↓ to highlight No**
- **Press Enter**



```
> npx
> create-vite . --template react-ts

|
o  Use rolldown-vite (Experimental)?:
|  No
|
o  Install with npm and start now?
|  No
|
o  Scaffolding project in C:\React Projects\hr-workflow-designer...
|
—  Done. Now run:

  npm install
  npm run dev


C:\React Projects\hr-workflow-designer>
```

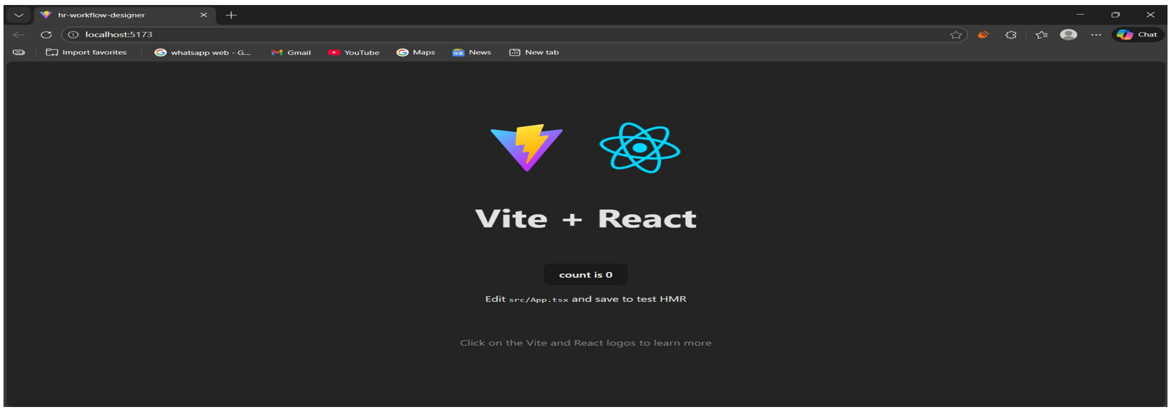
Step12: Now same Just Type this command in cmd: **npm install**

Step 13: Again, paste this Command in cmd: **npm run dev**



```
C:\React Projects\hr-workflow-designer>  npm install

added 177 packages, and audited 178 packages in 18s

45 packages are looking for funding
  run `npm fund` for details

found 0 vulnerabilities

C:\React Projects\hr-workflow-designer>  npm run dev

> hr-workflow-designer@0.0.0 dev
> vite


  VITE v7.2.7  ready in 432 ms

  ➜  Local:   http://localhost:5173/
  ➜  Network: use --host to expose
  ➜  press h + enter to show help
```

Step 14: Click on that Local link by Just **CTRL + click** link then it opens the **React Page** which we had created



Vite + React

Edit `src/App.tsx` and save to test HMR

Click on the Vite and React logos to learn more

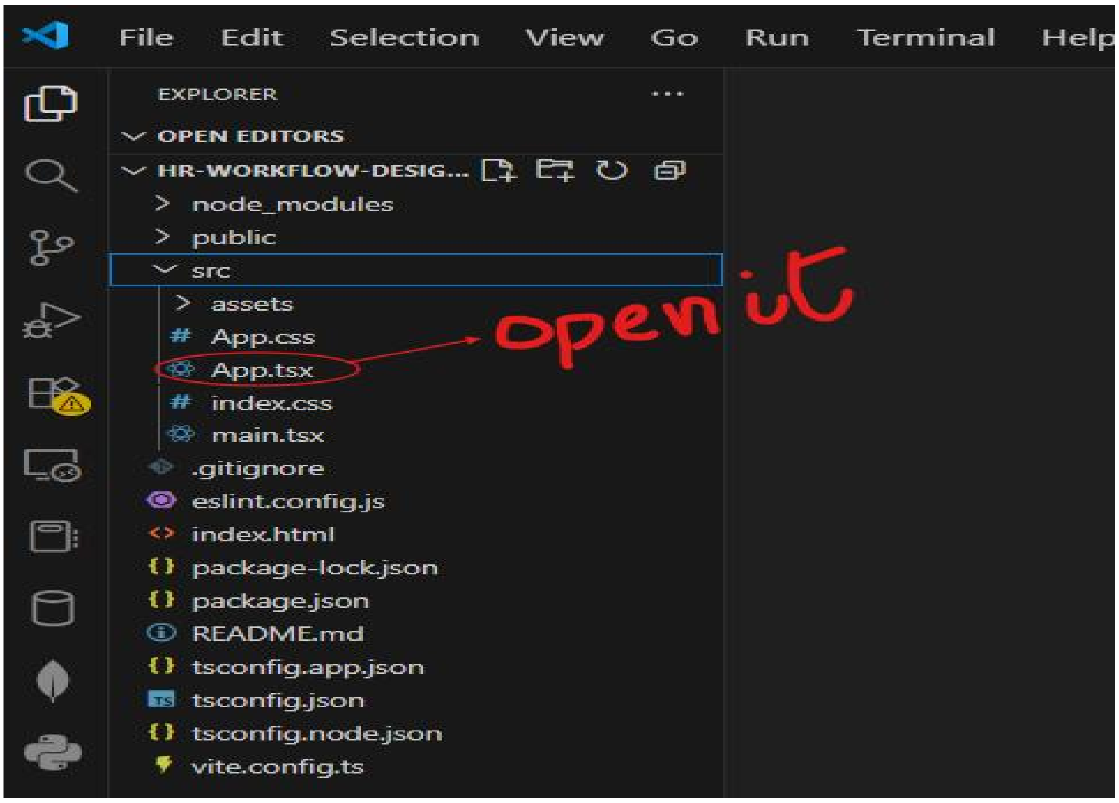
This means your React environment is now working perfectly.

Step 15: change this default Vite page into **"HR Workflow Designer"** so it feels like our project.

Step 16: Type this command in cmd: **code .**

and Press **Enter**

Step 17: Then it Opens **VS Code** and Shows as the Image Given Below and Open that **App.tsx** and Press **CTRL + A**, then delete it. and Just Type the New Code in it.



App.tsx Code:

```
// src/App.tsx
import React, {
  FC,
  useCallback,
  useState,
  ChangeEvent,
  useEffect,
} from "react";
import ReactFlow, {
  Background,
  Controls,
  MiniMap,
  Handle,
  Position,
  addEdge,
  useNodesState,
  useEdgesState,
} from "reactflow";
import type { Node, Edge, Connection } from "reactflow";
import "reactflow/dist/style.css";
type NodeKind = "start" | "task" | "approval" | "automated" | "end";
type NodeConfig = Record<string, any>;
type NodeData = {
  kind: NodeKind;
  label: string;
  config: NodeConfig;
};
type RFNode = Node<NodeData>;
// ---------- Custom node components ----------
type NodeComponentProps = { data: NodeData };
const baseNodeStyle: React.CSSProperties = {
  padding: "8px 12px",
  borderRadius: 8,
  fontSize: 12,
  minWidth: 160,
  textAlign: "center",
};
const StartNode: FC<NodeComponentProps> = ({ data }) => (
  <div
    style={{
      ...baseNodeStyle,
      borderRadius: 999,
      border: "2px solid #16a34a",
      background: "#dcfce7",
      fontWeight: 600,
    }}
  >
```

```
    <Handle type="source" position={Position.Bottom} />
    🚀 {data.label || "Start"}
  </div>
);
const TaskNode: FC<NodeComponentProps> = ({ data }) => (
  <div
    style={{
      ...baseNodeStyle,
      border: "1px solid #4b5563",
      background: "#111827",
      color: "white",
    }}
  >
    <Handle type="target" position={Position.Top} />
    <Handle type="source" position={Position.Bottom} />
    📝 {data.label || "Task"}
  </div>
);
const ApprovalNode: FC<NodeComponentProps> = ({ data }) => (
  <div
    style={{
      ...baseNodeStyle,
      border: "1px solid #6366f1",
      background: "#e0e7ff",
    }}
  >
    <Handle type="target" position={Position.Top} />
    <Handle type="source" position={Position.Bottom} />
    ✅ {data.label || "Approval"}
  </div>
);
const AutomatedNode: FC<NodeComponentProps> = ({ data }) => (
  <div
    style={{
      ...baseNodeStyle,
      border: "1px dashed #0ea5e9",
      background: "#e0f2fe",
    }}
  >
    <Handle type="target" position={Position.Top} />
    <Handle type="source" position={Position.Bottom} />
    🤖 {data.label || "Automated"}
  </div>
);
const EndNode: FC<NodeComponentProps> = ({ data }) => (
  <div
    style={{
      ...baseNodeStyle,
      borderRadius: 999,
      border: "2px solid #b91c1c",
      background: "#fee2e2",
      fontWeight: 600,
    }}
  >
    <Handle type="target" position={Position.Top} />
    🏁 {data.label || "End"}
  </div>
);
const nodeTypes = {
  start: StartNode,
  task: TaskNode,
  approval: ApprovalNode,
  automated: AutomatedNode,
  end: EndNode,
};
// ---------- Sidebar ----------
const sidebarButton: React.CSSProperties = {
  width: "100%",
  padding: "6px 10px",
  marginBottom: 8,
  borderRadius: 6,
  border: "1px solid #374151",
  background: "#111827",
  color: "white",
  fontSize: 12,
  cursor: "pointer",
  textAlign: "left",
};
const Sidebar: FC<{ onAddNode: (kind: NodeKind) => void }> = ({
  onAddNode,
}) => (
  <div
    style={{
      width: 220,
      padding: 12,
      borderRight: "1px solid #1f2933",
      background: "#020617",
      color: "white",
      height: "100vh",
      boxSizing: "border-box",
    }}
  >
    <h2 style={{ fontSize: 14, marginBottom: 12 }}>Node Palette</h2>
    <button style={sidebarButton} onClick={() => onAddNode("start")}>
      🚀 Start Node
    </button>
    <button style={sidebarButton} onClick={() => onAddNode("task")}>
      📝 Task Node
    </button>
    <button style={sidebarButton} onClick={() => onAddNode("approval")}>
      ☑️ Approval Node
    </button>
    <button style={sidebarButton} onClick={() => onAddNode("automated")}>
      🤖 Automated Step Node
    </button>
    <button style={sidebarButton} onClick={() => onAddNode("end")}>
      🏁 End Node
    </button>
  </div>
);
// ---------- Node Config Panel ----------
type ConfigPanelProps = {
  node: RFNode | null;
  onUpdate: (patch: Partial<NodeData["config"]> & { label?: string }) => void;
};
const labelCss: React.CSSProperties = {
  fontSize: 11,
  color: "#9ca3af",
  marginTop: 8,
  marginBottom: 4,
  display: "block",
};
const inputCss: React.CSSProperties = {
  width: "100%",
  padding: "6px 8px",
  borderRadius: 4,
  border: "1px solid #4b5563",
  background: "#020617",
  color: "white",
  fontSize: 12,
  boxSizing: "border-box",
};
const NodeConfigPanel: FC<ConfigPanelProps> = ({ node, onUpdate }) => {
  if (!node) {
    return (
      <div style={{ padding: 12, color: "#9ca3af", fontSize: 12 }}>
        Select a node to edit it.
      </div>
    );
  }
  const { kind, label, config } = node.data;
  const handleFieldChange =
    (field: string) =>
    (e: ChangeEvent<HTMLInputElement | HTMLTextAreaElement>) => {
      const value =
        e.currentTarget.type === "checkbox"
          ? (e.currentTarget as HTMLInputElement).checked
          : e.currentTarget.value;
      if (field === "label") {
        onUpdate({ label: value as string });
      } else {
        onUpdate({ [field]: value });
      }
    };
  return (
    <div style={{ padding: 12, color: "white", fontSize: 12 }}>
      <div style={{ fontSize: 14, fontWeight: 600, marginBottom: 8 }}>
        Node Details
      </div>
      <div style={{ fontSize: 11, marginBottom: 8, color: "#9ca3af" }}>
        Type: <strong>{kind.toUpperCase()}</strong>
      </div>
      {/* Common label */}
      <label style={labelCss}>Label</label>
      <input style={inputCss} value={label} onChange={handleFieldChange("label")} />
      {/* Type-specific fields */}
      {kind === "start" && (
        <>
          <label style={labelCss}>Start Title</label>
          <input
        style={inputCss}
        value={config.startTitle || ""}
        onChange={handleFieldChange("startTitle")}
      />
      <label style={labelCss}>Metadata (key=value, comma separated)</label>
      <textarea
        style={{ ...inputCss, minHeight: 60 }}
        value={config.metadata || ""}
        onChange={handleFieldChange("metadata")}
      />
    </>
  )}
  {kind === "task" && (
    <>
      <label style={labelCss}>Title</label>
      <input
        style={inputCss}
        value={config.title || ""}
        onChange={handleFieldChange("title")}
      />
      <label style={labelCss}>Description</label>
      <textarea
        style={{ ...inputCss, minHeight: 60 }}
        value={config.description || ""}
        onChange={handleFieldChange("description")}
      />
      <label style={labelCss}>Assignee</label>
      <input
        style={inputCss}
        value={config.assignee || ""}
        onChange={handleFieldChange("assignee")}
      />
      <label style={labelCss}>Due Date</label>
      <input
```

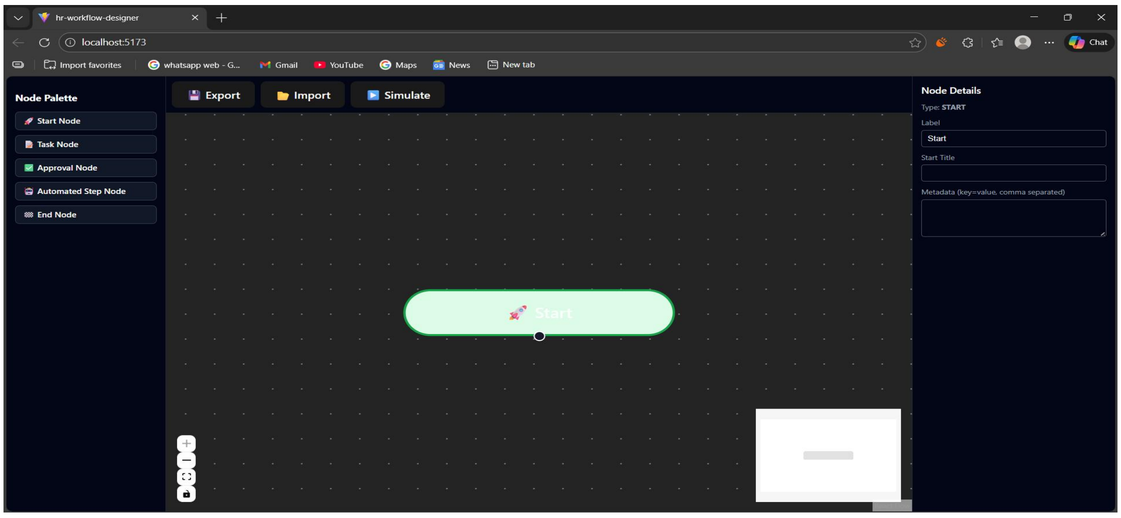
```
        type="date"
        style={inputCss}
        value={config.dueDate || ""}
        onChange={handleFieldChange("dueDate")}
      />
      <label style={labelCss}>Custom Fields (key=value)</label>
      <textarea
        style={{ ...inputCss, minHeight: 60 }}
        value={config.customFields || ""}
        onChange={handleFieldChange("customFields")}
      />
    </>
  )}
  {kind === "approval" && (
    <>
      <label style={labelCss}>Title</label>
      <input
        style={inputCss}
        value={config.title || ""}
        onChange={handleFieldChange("title")}
      />
      <label style={labelCss}>Approver Role</label>
      <input
        style={inputCss}
        value={config.approverRole || ""}
        onChange={handleFieldChange("approverRole")}
        placeholder="Manager, HRBP, Director..."
      />
      <label style={labelCss}>Auto-approve Threshold</label>
      <input
        type="number"
        style={inputCss}
        value={config.threshold ?? ""}
        onChange={handleFieldChange("threshold")}
    />
  </>
)}
{kind === "automated" && (
  <>
    <label style={labelCss}>Title</label>
    <input
      style={inputCss}
      value={config.title || ""}
      onChange={handleFieldChange("title")}
    />
    <label style={labelCss}>Action ID</label>
    <input
      style={inputCss}
      value={config.actionId || ""}
      onChange={handleFieldChange("actionId")}
      placeholder="send_email, generate_doc..."
    />
    <label style={labelCss}>Action Params (JSON / key=value)</label>
    <textarea
      style={{ ...inputCss, minHeight: 60 }}
      value={config.params || ""}
      onChange={handleFieldChange("params")}
    />
  </>
)}
{kind === "end" && (
  <>
    <label style={labelCss}>End Message</label>
    <input
      style={inputCss}
      value={config.message || ""}
      onChange={handleFieldChange("message")}
    />
```

```
          <label style={labelCss}>
            <input
              type="checkbox"
              style={{ marginRight: 6 }}
              checked={!!config.summary}
              onChange={handleFieldChange("summary")}
            />
            Include in summary
          </label>
        </>
      )}
    </div>
  );
};
// ---------- Initial nodes ----------
const initialNodes: RFNode[] = [
  {
    id: "1",
    type: "start",
    position: { x: 350, y: 80 },
    data: {
      kind: "start",
      label: "Start",
      config: {},
    },
  },
];
// ---------- Main App ----------
const App: FC = () => {
  const [nodes, setNodes, onNodesChange] = useNodesState<RFNode[]>(initialNodes);
  const [edges, setEdges, onEdgesChange] = useEdgesState<Edge[]>([]);
  const [selectedId, setSelectedId] = useState<string>("1");
  const onConnect = useCallback(
    (params: Edge | Connection) => setEdges((eds) => addEdge(params, eds)),
  []
);
const labelByKind: Record<NodeKind, string> = {
  start: "Start",
  task: "Task",
  approval: "Approval",
  automated: "Automated Step",
  end: "End",
};
const handleAddNode = (kind: NodeKind) => {
  setNodes((prev) => {
    const newNode: RFNode = {
      id: crypto.randomUUID(),
      type: kind,
      position: { x: 350, y: 80 + prev.length * 80 },
      data: {
        kind,
        label: labelByKind[kind],
        config: {},
      },
    };
    return [...prev, newNode];
  });
};
const handleNodeClick = (_: React.MouseEvent, node: RFNode) => {
  setSelectedId(node.id);
};
const selectedNode = nodes.find((n) => n.id === selectedId) || null;
const updateSelectedConfig: ConfigPanelProps["onUpdate"] = (patch) => {
  if (!selectedNode) return;
  const id = selectedNode.id;
  setNodes((prev) =>
    prev.map((n) => {
      if (n.id !== id) return n;
      const newLabel =
        typeof patch.label === "string" ? patch.label : n.data.label;
      const { label, ...rest } = patch;
      return {
        ...n,
        data: {
          ...n.data,
          label: newLabel,
          config: {
            ...n.data.config,
            ...rest,
          },
        },
      };
    })
  );
};
// ---------- Delete selected node with Delete/Backspace ----------
const handleKeyDown = useCallback(
  (e: KeyboardEvent) => {
    if (!selectedId) return;
    if (e.key === "Delete" || e.key === "Backspace") {
      setNodes((prev) => prev.filter((n) => n.id !== selectedId));
      setEdges((prev) =>
        prev.filter(
          (edge) =>
            edge.source !== selectedId && edge.target !== selectedId
        )
      );
      setSelectedId("");
    }
  },
  [selectedId, setNodes, setEdges]
);
```

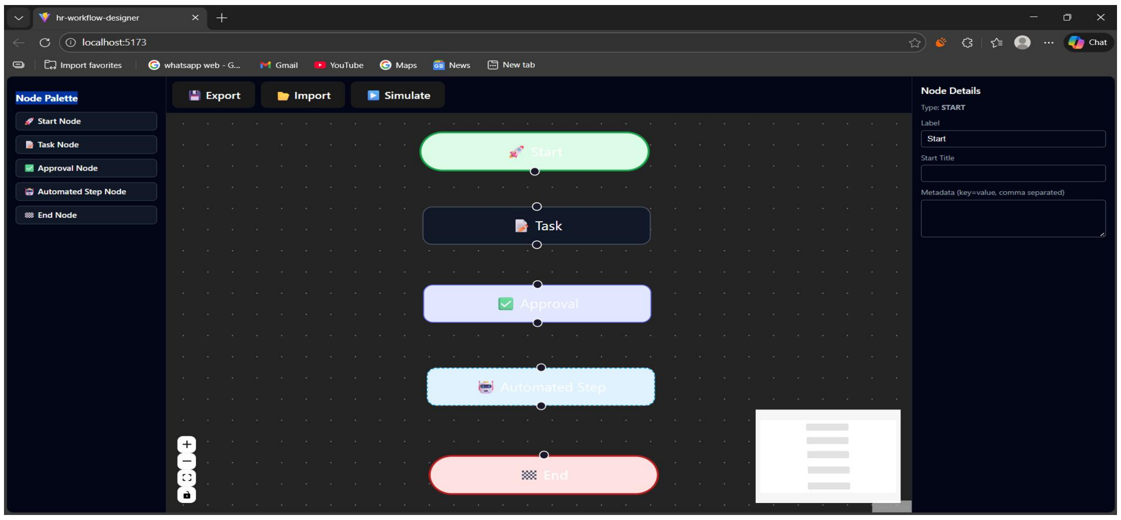
```
useEffect(() => {
  window.addEventListener("keydown", handleKeyDown);
  return () => window.removeEventListener("keydown", handleKeyDown);
}, [handleKeyDown]);

// ---------- Export / Import / Simulate ----------
const handleExport = () => {
  const workflow = {
    nodes: nodes.map((n) => ({
      id: n.id,
      type: n.data.kind,
      label: n.data.label,
      config: n.data.config,
      position: n.position,
    })),
    edges,
  };
  navigator.clipboard.writeText(JSON.stringify(workflow, null, 2));
  alert("Workflow JSON copied to clipboard.");
};
const handleImport = () => {
  const json = prompt("Paste workflow JSON:");
  if (!json) return;
  try {
    const parsed = JSON.parse(json);
    setNodes(
      parsed.nodes.map((n: any) => ({
        id: n.id,
        type: n.type,
        position: n.position,
        data: {
          kind: n.type as NodeKind,
          label: n.label,
          config: n.config || {},
        },
      }))
    );
    setEdges(parsed.edges || []);
    alert("Workflow loaded.");
  } catch (err) {
    alert("Invalid JSON. Please check and try again.");
  }
};
const handleSimulate = () => {
  const steps = [...nodes]
    .sort((a, b) => a.position.y - b.position.y)
    .map((n) => {
      const cfg = n.data.config || {};
      switch (n.data.kind) {
        case "start":
          return `🚀 Start: ${n.data.label}`;
        case "task":
          return `📝 Task: ${n.data.label} — Assignee: ${
            cfg.assignee || "Unassigned"
          }`;
        case "approval":
          return `✅ Approval: ${n.data.label} — Role: ${
            cfg.approverRole || "N/A"
          }`;
        case "automated":
          return `🤖 Automated: ${n.data.label} — Action: ${
            cfg.actionId || "none"
          }`;
        case "end":
          return `🏁 End: ${n.data.label}`;
        default:
          return n.data.label;
      }
    });
  alert(steps.join("\n"));
};
return (
  <div style={{ display: "flex", width: "100vw", height: "100vh" }}>
    <Sidebar onAddNode={handleAddNode} />
    <div
      style={{
        flex: 1,
        display: "flex",
        flexDirection: "column",
      }}
    >
      {/* Toolbar */}
      <div
        style={{
          padding: 8,
          display: "flex",
          gap: 8,
          borderBottom: "1px solid #1f2933",
          background: "#020617",
        }}
      >
        <button onClick={handleExport}>💾 Export</button>
        <button onClick={handleImport}>📂 Import</button>
        <button onClick={handleSimulate}>▶️ Simulate</button>
      </div>
      {/* Canvas */}
      <div style={{ flex: 1 }}>
        <ReactFlow
          nodes={nodes}
          edges={edges}
          nodeTypes={nodeTypes}
          onNodesChange={onNodesChange}
          onEdgesChange={onEdgesChange}
          onConnect={onConnect}
          onNodeClick={handleNodeClick}
          fitView
        >
          <Background />
          <MiniMap pannable zoomable />
          <Controls />
        </ReactFlow>
      </div>
    </div>
    <div
      style={{
        width: 280,
        borderLeft: "1px solid #1f2933",
        background: "#020617",
        height: "100vh",
        boxSizing: "border-box",
      }}
    >
      <NodeConfigPanel node={selectedNode} onUpdate={updateSelectedConfig} />
    </div>
  </div>
 );
};
export default App;
```

Step 18: Again, go to cmd and type the command: **npm run dev**

and press **Enter** and Click on that Local link **CTRL + click** on it then it opens your app is running and showing **HR Workflow Designer**.

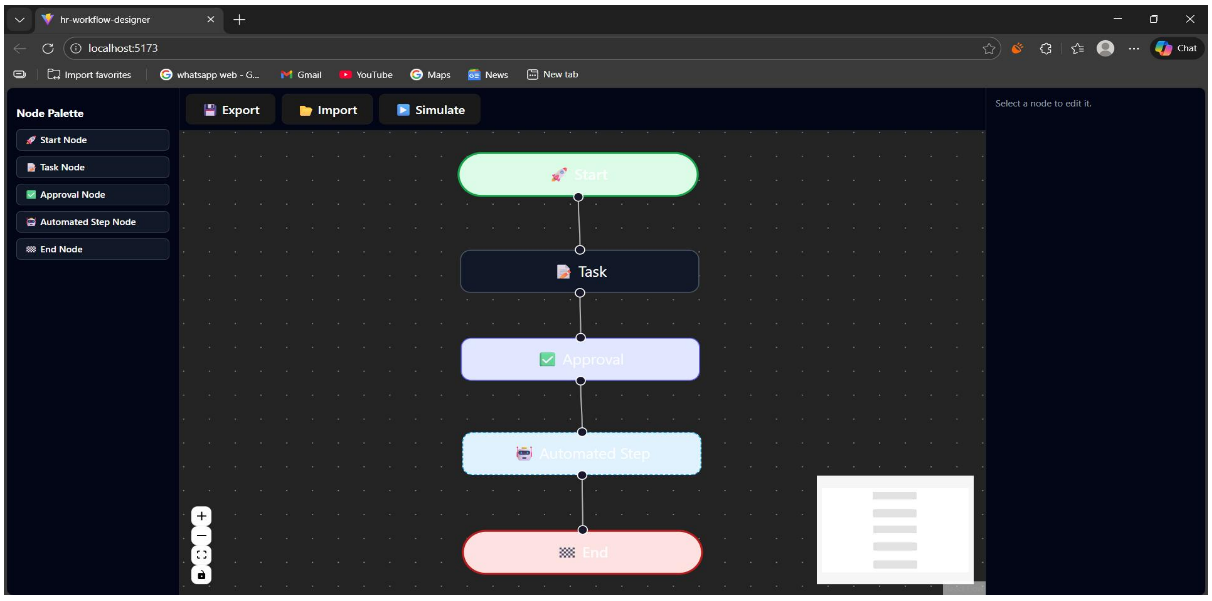


Step 19: Adding Up Each Nodes Which we can See it under **Node Palette**

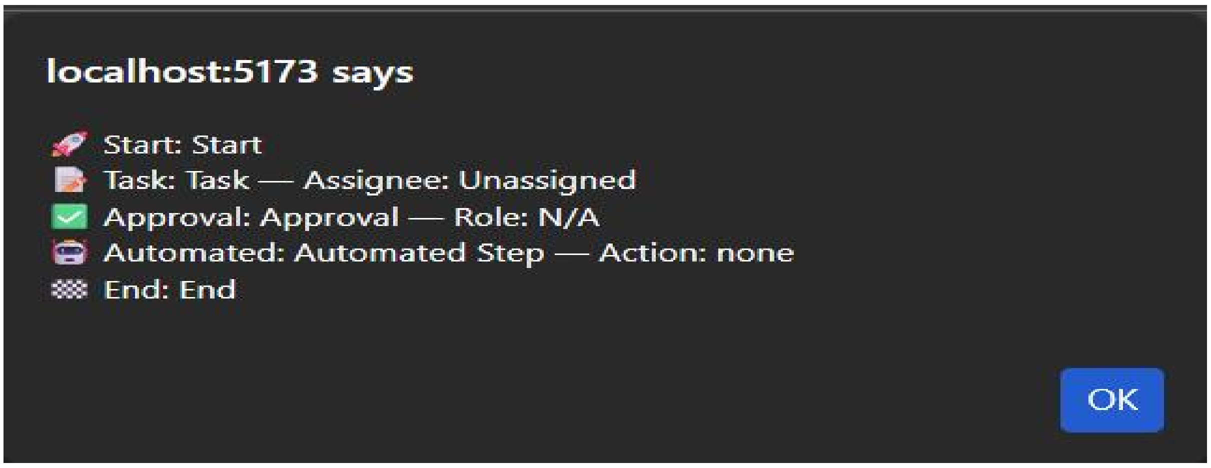


Step 20: After Adding all **Node Palette,** we need connect with each other

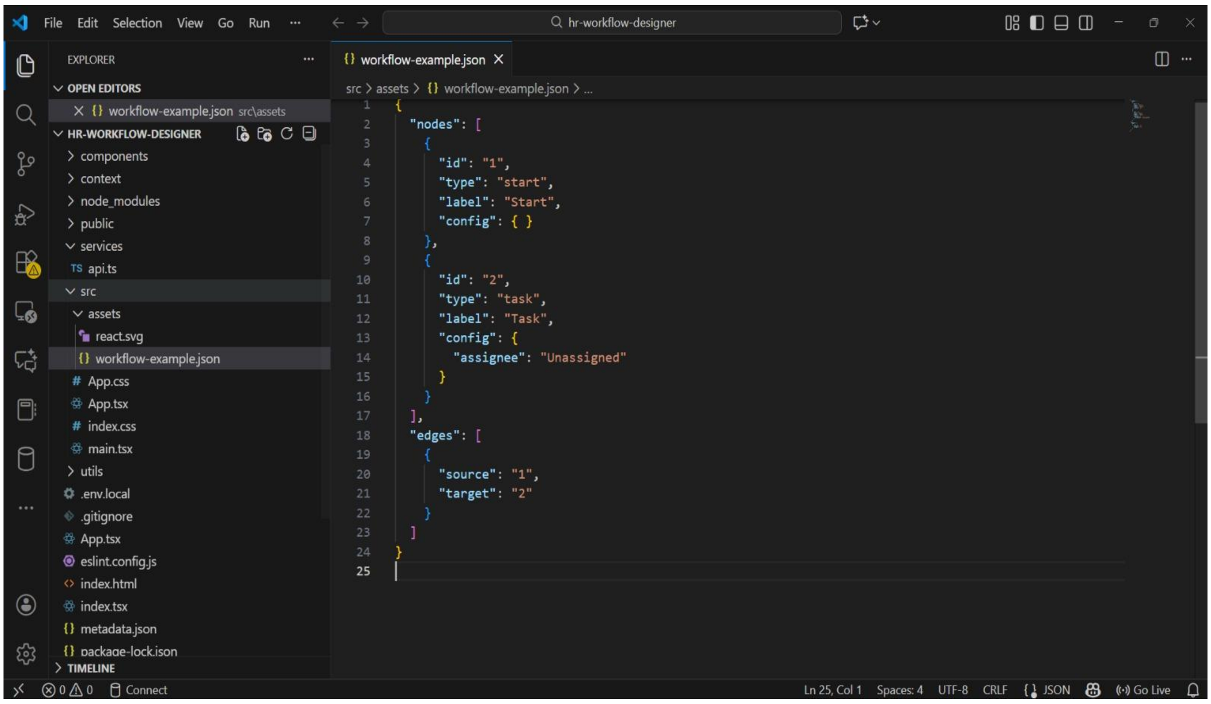
Like: Start → Task → Approval → Automated → End



Step 21: After Connecting with each other click on that **Simulate**



```
localhost:5173 says

🚀 Start: Start
📝 Task: Task — Assignee: Unassigned
✅ Approval: Approval — Role: N/A
🤖 Automated: Automated Step — Action: none
🏁 End: End
```



```json
{
  "nodes": [
    {
      "id": "1",
      "type": "start",
      "label": "Start",
      "config": { }
    },
    {
      "id": "2",
      "type": "task",
      "label": "Task",
      "config": {
        "assignee": "Unassigned"
      }
    }
  ],
  "edges": [
    {
      "source": "1",
      "target": "2"
    }
  ]
}
```

4. Project Structure:

```
🗀 Desktop
└─ 🗀 hr-workflow-designer
   |
   ├─ 🗀 node_modules
   |
   ├─ 🗀 public
   |   └─ 🗎 index.html
   |
   ├─ 🗀 src
```

```
│   │
│   ├─ 📄 App.tsx
│   ├─ 📄 App.css
│   ├─ 📄 index.tsx
│   ├─ 📄 index.css
│   ├─ 📄 main.tsx
│
│   ├─ 📁 assets
│   │   ├─ 📄 react.svg
│   │   └─ 📄 workflow-example.json  ← ✓ Exported JSON file
│
│   ├─ 📁 components
│   │   └─ (optional files)
│
│   ├─ 📁 context
│   │   └─ (optional files)
│
│   ├─ 📁 services
│   │   └─ 📄 api.ts
│
│   └─ 📁 utils
│      └─ (optional helper files)
│
├─📄 . env.local (optional)
├─ 📄 . gitignore
├─ 📄 package.json
├─ 📄 package-lock.json
├─ 📄 tsconfig.json
├─ 📄 tsconfig.app.json
├─ 📄 tsconfig.node.json
├─ 📄 vite.config.ts
└─ 📄 README.md
```

5. CONCLUSION:

The HR Workflow Designer successfully demonstrates how complex HR processes can be visualized and managed using a modern web-based interface.
By leveraging ReactFlow and TypeScript, the project provides:

- Clear workflow visualization
- User-friendly configuration
- Automated workflow export/import
- Execution simulation

This tool can be extended for real-world HR systems such as employee onboarding, leave approval processes, and document automation workflows.
The assignment showcases practical implementation of UI design, state management, and workflow logic.

Done By:

Name: PINISETTI GOVARDHAN,

Reg. No.: RA2211056010107,

Dept. Name: CSE – DATA SCIENCE,

Section: AF-2,

Company Name: TREDENCE,

Role: FULL STACK DEVELOPER INTERN ASSIGNMENT.