# Step 1: Initialize the Vite React project with dependencies

1. In VS Code, open the integrated terminal at your frontend-ui folder.
2. Run this command to bootstrap a React + TypeScript Vite app right into this folder:

npm create vite@latest . -- --template react-ts

**What this does:**

* Creates all the boilerplate files (package.json, tsconfig.json, vite.config.ts, src/, etc.)
* Installs React, React DOM, Vite dev server, and TypeScript setup
* Sets us up to use .tsx for components (matching our src/nodes/index.tsx and types/\*.ts references)

Once that finishes, you’ll see a complete folder structure under frontend-ui matching a standard Vite + React-TS project.

A screenshot of a computer program

AI-generated content may be incorrect.

Because I already had a workflow\_project\_readme.md, I was warned that the folder was not empty. I used the arrow keys to move down to the Ignore files option and clicked enter.

Next step is to run the line “npm install”. We do not need to run “npm run dev” yet. We have more command line things to do.

A screen shot of a computer

AI-generated content may be incorrect.

After this, the files and folders on the left look like this (green are new to the git repo, white are unchanged)

A screenshot of a computer

AI-generated content may be incorrect.

If you do run the app so far, you will get the default vite / react page.

Eg. Typing in to the terminal “npm run dev” and clicking enter, you get this:

A computer screen with white text

AI-generated content may be incorrect.

The cyan is the URL of the local Vite + React app. To open that URL, hold down CTRL and click it.  
This is the default app that you will see:  
A screenshot of a computer

AI-generated content may be incorrect.

Close the tab for now and in Visual Studio Code, type q, then press enter, to stop the localhost:5173 app server.

A black screen with white text

AI-generated content may be incorrect.

We still need to install the special technology called React-Flow library, which exposes the <ReactFlow> component that under the hood manages an interactive canvas. We need that to display nodes etc.

In terminal, type:

npm install reactflow

A computer screen with text and numbers

AI-generated content may be incorrect.

## Summary

 **npm create vite@latest . -- --template react-ts**  
Bootstraps a React + TypeScript Vite app, creating all the boilerplate (package.json, tsconfig.json, vite.config.ts, src/, etc.) so you can write .tsx components immediately.

 **npm install**  
Installs the core dependencies Vite generated (React, React DOM, TypeScript, Vite plugins).

 **npm install reactflow**  
Adds the React-Flow library, which provides the <ReactFlow /> canvas and node primitives we’ll use.

 **npm run dev**  
Launches Vite’s development server (usually at http://localhost:5173), verifying everything compiles and loads correctly.

# **Step 2: Add the Canvas component** and wire in our first dummy node

Create a new file at src/components/Canvas.tsx with the following content:

|  |
| --- |
| // src/components/Canvas.tsx  // <<canvas-imports>>  import React from 'react';  import ReactFlow, {  Node,  Edge,  useNodesState,  useEdgesState,  NodeChange,  EdgeChange,  OnNodesChange,  OnEdgesChange,  OnConnect,  } from 'reactflow';  import 'reactflow/dist/style.css';  /\*\*  \* A simple Canvas component that renders a React-Flow canvas  \* with one dummy node.  \*/  export default function Canvas() {  // <<initial-nodes>>  // Define a single dummy node to show on the canvas:  const initialNodes: Node[] = [  {  id: '1',  type: 'default', // default node type (a simple box)  position: { x: 0, y: 0 }, // top-left corner of canvas  data: { label: 'Dummy Node' }, // the text label inside the node  },  ];  // <<initial-edges>>  // No edges (connections) yet:  const initialEdges: Edge[] = [];  // <<nodes-state>>  // useNodesState gives us:  // • nodes: current array of Node objects  // • setNodes: a setter to replace them (not usually needed directly)  // • onNodesChange: a handler we pass to <ReactFlow> to auto-update positions, etc.  const [nodes, , onNodesChange] = useNodesState(initialNodes);  // <<edges-state>>  // useEdgesState similarly manages edges for you:  const [edges, , onEdgesChange] = useEdgesState(initialEdges);  // <<on-connect-handler>>  // If you eventually want to connect nodes with edges, React-Flow will call this:  const onConnect: OnConnect = (connection) => {  // In the future: add new edge to the list  // (for now, we’re not drawing edges, so no-op)  };  // <<canvas-render>>  // Return the ReactFlow component inside a full-viewport wrapper:  return (  <div style={{ width: '100%', height: '100vh' }}>  <ReactFlow  nodes={nodes}  edges={edges}  onNodesChange={onNodesChange as OnNodesChange}  onEdgesChange={onEdgesChange as OnEdgesChange}  onConnect={onConnect}  fitView={true} // automatically zoom/pan so all nodes are visible  />  </div>  );  } |