**RideSync**

**1. Project Initialization**

* **Command Used:**  
  npm create vite@latest ridesync-frontend -- --template react-ts
* **Why:**  
  Vite is a fast build tool for modern web apps. The React + TypeScript template gives you a solid foundation for scalable, type-safe development.
* **What Happened:**  
  Vite scaffolded a new project folder ([ridesync-frontend](vscode-file://vscode-app/c:/Users/Gayani%20Gunasekara/AppData/Local/Programs/Microsoft%20VS%20Code/resources/app/out/vs/code/electron-browser/workbench/workbench.html)) with the following structure:
  + src/ (source code)
  + public/ (static assets)
  + index.html (main HTML file)
  + Configuration files (vite.config.ts, tsconfig.json, etc.)

**2. Dependency Installation**

* **Command Used:**  
  npm install
* **Why:**  
  Installs all dependencies listed in [package.json](vscode-file://vscode-app/c:/Users/Gayani%20Gunasekara/AppData/Local/Programs/Microsoft%20VS%20Code/resources/app/out/vs/code/electron-browser/workbench/workbench.html) (React, ReactDOM, TypeScript, Vite, etc.).
* **What Happened:**
  + node\_modules/ folder was created to store installed packages.
  + package-lock.json was generated to lock dependency versions.

**3. Starting the Development Server**

* **Command Used:**  
  npm run dev
* **Why:**  
  Launches Vite’s development server for live preview and hot reload.
* **What Happened:**
  + The app became available at http://localhost:5173/ (or another port if busy).
  + Any changes in src/ automatically update the browser.

**4. Folder Structure Organization**

* **Why:**  
  Organizing code into logical folders makes the project scalable and maintainable.
* **What Happened:**  
  You created and organized these folders inside src/:
  + components/ — For reusable UI components (e.g., [Layout.tsx](vscode-file://vscode-app/c:/Users/Gayani%20Gunasekara/AppData/Local/Programs/Microsoft%20VS%20Code/resources/app/out/vs/code/electron-browser/workbench/workbench.html))
  + pages/ — For page-level components (e.g., [HomePage.tsx](vscode-file://vscode-app/c:/Users/Gayani%20Gunasekara/AppData/Local/Programs/Microsoft%20VS%20Code/resources/app/out/vs/code/electron-browser/workbench/workbench.html))
  + services/ — For API calls and business logic
  + hooks/ — For custom React hooks
  + [assets](vscode-file://vscode-app/c:/Users/Gayani%20Gunasekara/AppData/Local/Programs/Microsoft%20VS%20Code/resources/app/out/vs/code/electron-browser/workbench/workbench.html) — For images and static files

**5. Tailwind CSS Integration**

* **Commands/Steps Used:**  
  Installed Tailwind and PostCSS, then configured them.
* **Why:**  
  Tailwind CSS provides utility-first styling, speeding up UI development.
* **What Happened:**
  + tailwind.config.js and postcss.config.js were created.
  + You can now use Tailwind classes in your JSX (className="...").

**6. React Router Setup**

* **Why:**  
  Enables client-side routing for multi-page navigation.
* **What Happened:**
  + In [App.tsx](vscode-file://vscode-app/c:/Users/Gayani%20Gunasekara/AppData/Local/Programs/Microsoft%20VS%20Code/resources/app/out/vs/code/electron-browser/workbench/workbench.html), you set up routing using [BrowserRouter](vscode-file://vscode-app/c:/Users/Gayani%20Gunasekara/AppData/Local/Programs/Microsoft%20VS%20Code/resources/app/out/vs/code/electron-browser/workbench/workbench.html), [Routes](vscode-file://vscode-app/c:/Users/Gayani%20Gunasekara/AppData/Local/Programs/Microsoft%20VS%20Code/resources/app/out/vs/code/electron-browser/workbench/workbench.html), and [Route](vscode-file://vscode-app/c:/Users/Gayani%20Gunasekara/AppData/Local/Programs/Microsoft%20VS%20Code/resources/app/out/vs/code/electron-browser/workbench/workbench.html).
  + The main layout ([Layout.tsx](vscode-file://vscode-app/c:/Users/Gayani%20Gunasekara/AppData/Local/Programs/Microsoft%20VS%20Code/resources/app/out/vs/code/electron-browser/workbench/workbench.html)) wraps your pages.
  + The homepage ([HomePage.tsx](vscode-file://vscode-app/c:/Users/Gayani%20Gunasekara/AppData/Local/Programs/Microsoft%20VS%20Code/resources/app/out/vs/code/electron-browser/workbench/workbench.html)) is shown at /.
  + Any unknown route redirects to / (using <Navigate />).

**7. Configuration Files**

* **Why:**  
  These files control how your project is built and linted.
* **What Happened:**
  + vite.config.ts — Vite configuration
  + tsconfig.json — TypeScript settings
  + eslint.config.js — Linting rules
  + .gitignore — Files to exclude from Git

**8. Result**

* **Why:**  
  All these steps set up a modern, maintainable, and fast development environment.
* **What Happened:**
  + You have a working React + TypeScript app with hot reload and Tailwind CSS.
  + The folder structure is ready for scalable development.
  + You can now focus on building features and UI inside src/.

**Summary:**  
You used Vite to quickly scaffold a React + TypeScript project, installed dependencies, set up Tailwind CSS for styling, organized your codebase into logical folders, and configured routing for navigation. The result is a solid foundation for building your RideSync app efficiently.

**Splash Screen Development**

**1. Installed Animation & Icon Libraries**

**Command:**

npm install framer-motion

**How & Why:**

* You opened the terminal in your project directory and ran this command to add **Framer Motion** to your project.
* This allowed you to use React components from Framer Motion, such as <motion.div>, <motion.img>, etc.
* You used these components to apply animations like **fade-in/out**, **scaling**, and **sliding** for your logo and images.
* By wrapping elements with Framer Motion, you defined animation **variants** and **transition properties** to control the timing, easing, and motion effects.

**Command:**

npm install @heroicons/react

**How & Why:**

* You installed **Heroicons**, a library of SVG icons for React, to easily add professional-looking icons.
* After installing, you imported icons directly into your splash screen component, for example:

import { CheckCircleIcon } from '@heroicons/react/solid';

* You used these icons to enhance visual appeal without manually adding SVG files, keeping the code clean and maintainable.

**2. Splash Screen Development**

**How You Did It:**

* **Component Creation:**
  + Created a new file SplashScreen.tsx in src/pages/.
  + Defined a functional React component with JSX that renders your logo, icons, and optional background images.
* **Animations:**
  + Wrapped key elements (logo, icons) in <motion.div> or <motion.img> components.
  + Defined **initial**, **animate**, and **exit** states for elements to control how they appear and disappear.
  + Example:
* <motion.img
* src={logo}
* initial={{ opacity: 0, scale: 0.8 }}
* animate={{ opacity: 1, scale: 1 }}
* transition={{ duration: 1.5 }}
* />
* **Icons Integration:**
  + Imported Heroicons and added them next to text or images for visual enhancement.
  + Styled them with Tailwind CSS or custom CSS to match your splash screen’s design.
* **Centering & Styling:**
  + Used CSS Flexbox or Grid to center all elements horizontally and vertically:
* .splash-container {
* display: flex;
* justify-content: center;
* align-items: center;
* height: 100vh;
* }
  + Adjusted image sizes using width, height, or responsive units like vw, vh, or % to make them look proportional.
* **Responsive Design:**
  + Used responsive CSS or Tailwind classes to ensure images and icons scale correctly on all devices.
  + Tested on multiple screen sizes and adjusted breakpoints for mobile, tablet, and desktop layouts.
  + Ensured text and icons maintain readability and spacing across devices.
* **Integration with App Flow:**
  + In App.tsx, you added a useState to control splash screen visibility.
  + Used setTimeout in useEffect to display the splash screen for a few seconds, then navigate to the homepage.
* const [showSplash, setShowSplash] = useState(true);
* useEffect(() => {
* const timer = setTimeout(() => setShowSplash(false), 2500);
* return () => clearTimeout(timer);
* }, []);

**3. Additional Steps**

* Ran npm install to ensure all dependencies were correctly installed and up-to-date.
* Added/updated splash screen assets: logos, background images, icons.
* Edited CSS or component styles to maintain **centering**, **scalability**, and **responsive layout**.
* Tested animations and responsiveness on multiple devices and screen sizes.
* Optimized code structure to allow easy future updates and enhancements.

**4. Result**

* The RideSync project now has a **modern, animated splash screen** with smooth transitions and professional icons.
* The splash screen is **fully responsive** and scales well on all devices.
* The project is prepared for further UI improvements with a clean and maintainable codebase.

**Summary**

Today, you enhanced the RideSync web app by:

1. Installing **Framer Motion** and applying animations to elements.
2. Installing **Heroicons** and integrating them as visual enhancements.
3. Developing a **centered splash screen** with properly sized and aligned elements.
4. Ensuring the splash screen is **responsive and scalable** across devices.
5. Integrating the splash screen into the app flow to display before the homepage.

These steps created a polished first impression for users and set the foundation for future UI improvements.