|  |
| --- |
| **ChessBoard Game**  *“The chessboard displays an 8×8 grid with alternating colors, and clicking a square changes its color using Redux to manage the board’s state dynamically.”*  **Developed By: Gaurvi Paneri**  13 August, 2025.  BrainyBeam Placement Task |

# **Table of Contents**

1. **Introduction** *Overview and purpose of the ChessBoard App*
2. **Objective** *What app aims to achieve and why*
3. **Features** *Description of key functionalities*
4. **Technology Used** *Tools and libraries used in the project*
5. **Application Architecture** *Explanation of frontend structure and data flow*
6. **Implementation Details** *Key code logic, components, and state management*
7. **Usage Instructions** *How to run and use the app locally*
8. **Screenshots** *Screenshots of the output*
9. **Challenges and Solutions** *Challenges that were faced along with their solutions*
10. **Conclusion** *Summary of the project and learnings*
11. **References** *Resources and documentation referred to during development*
12. **Introduction**

The ChessBoard Game is a simple web application that renders an 8×8 grid with alternating black and white squares, mimicking a traditional chessboard pattern. The application leverages Redux for state management, tracking which squares have been clicked. When a user clicks on a square, the app updates the Redux store to reflect the change, and the square’s color dynamically changes based on predefined CSS logic: white squares turn yellow, and black squares turn red.

This project demonstrates effective use of React for UI rendering combined with Redux to manage and update application state in response to user interactions.

1. **Objective**

The Objectives of this project were as follows:

* To create and display an 8×8 grid resembling a chessboard, with alternating black and white squares.
* To track and manage the states of squares clicked by users using Redux.
* To dynamically change the color of clicked squares based on their original color: black squares change to red, and white squares change to yellow.

1. **Features**

The key Features of ChessBoard Game includes:

* A visually accurate 8×8 chessboard grid with alternating black and white squares.
* Efficient state management using Redux to keep track of user interactions with the board.
* Dynamic color transformation of squares upon clicking — black squares transition to red, and white squares switch to yellow, providing immediate visual feedback.

1. **Technologies Used**

* **React with Vite:** Enables fast and efficient development with a modern build tool and a reactive UI library.
* **Redux Toolkit & Redux:** Provides robust and scalable state management for tracking and updating the board’s state.
* **CSS:** Implements the styling and color-changing logic for the chessboard squares upon user interaction

1. **Application Architecture**

**1.Folder Structure**

**ChessBoard\_Code**

**├──src**

**│ ├── components**

**│ └── ChessBoard.jsx→***Renders chessboard and handles click events*

**│ ├── features**

**│ └── BoardSlice.jsx→***stores the board state and color change logic*

**│ ├── store**

**│ └── Store.jsx→***Combines the Slice and provides redux stor*

**│ ├──App.css→***For styling main app layout*

**│ ├── App.jsx→***Consists of routes that integrated the ChessBoard*

**└── main.jsx→***Application entry point*

**├──public**

**├──index.html**

**└──package.json**

**2. Data Flow**

* **ChessBoard.jsx**
  + Renders the 8×8 chessboard UI.
  + Listens for click events on each square.
  + When a square is clicked, it **dispatches an action** to the Redux store.
* **boardSlice.jsx**
  + Receives the dispatched action.
  + Updates the state (which square was clicked, and its new color logic).
  + Contains the reducer logic for changing white → yellow and black → red.
* **store.jsx**
  + Holds the central Redux store that contains **boardSlice** state.
  + Makes the state available to the entire app via **<Provider>**.
* **Re-render**
  + After the state changes, **React re-renders ChessBoard.jsx** with the updated state from Redux.
  + CSS logic applies new colors based on the updated state.

**3. How Components Interact**

* **The user clicks** on a square in the chessboard.
* **ChessBoard.jsx** dispatches an action to Redux using useDispatch.
* **boardSlice.jsx** receives the action and updates the state (color change logic applied).
* **store.jsx** stores the updated board state centrally.
* **ChessBoard.jsx** reads the latest state from Redux using useSelector.
* The **UI re-renders**, showing the updated colors for the clicked squares.

1. **Implementation Details**

The application is a replica of a chessboard with an 8×8 grid, designed to demonstrate how Redux manages state and tracks user actions. It simulates the process of clicking on individual squares and updating their colors based on their original color. When a user clicks on a white square, it changes to yellow, and when a black square is clicked, it changes to red. This updated state is stored locally in Redux and is maintained until the page is refreshed. The project serves as a simple yet effective example of how Redux can be used to handle state changes and user interactions in a React application.

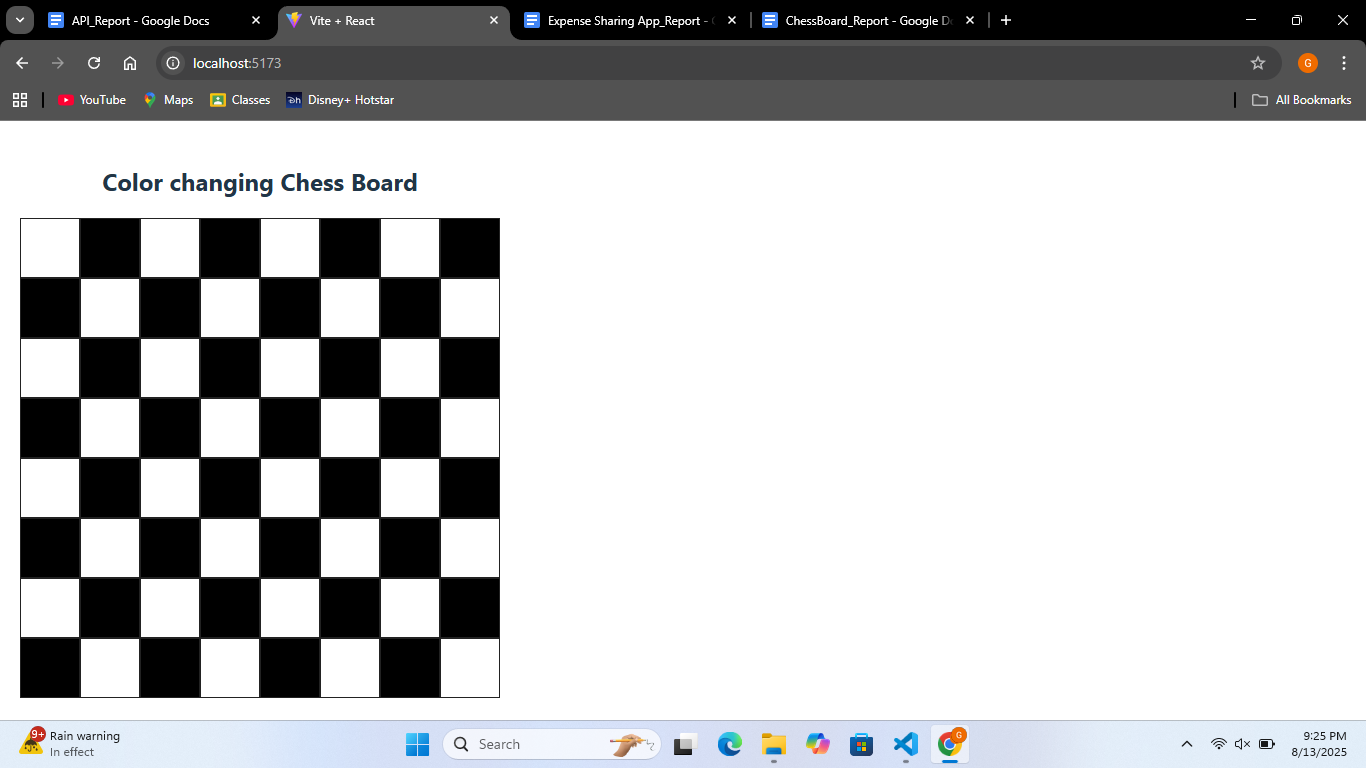
1. **Usage Instruction**

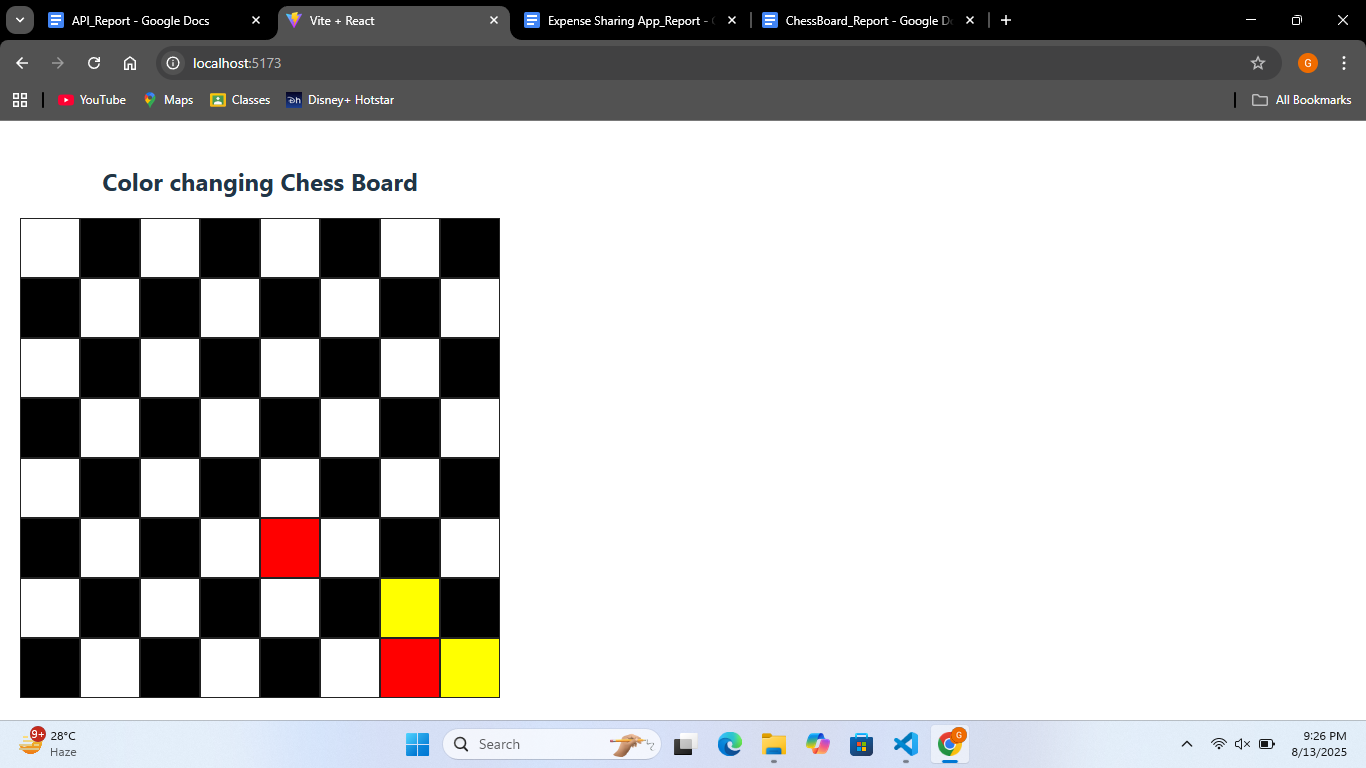
### **Running the App Locally**

### **Prerequisites:**

* + Make sure you have **Node.js (v16 or higher)** installed on your system.
  + A modern web browser (Chrome, Firefox, Edge, etc.).
* **Installation:**
  + Clone the project repository to your local machine:  
     **git clone React\_Task\_1\_GaurviPaneri**
  + Navigate to the project folder:  
    **cd React\_Task\_1\_GaurviPaneri/code/ChessBoard\_Code**
  + Install dependencies:  
    **npm install**
* **Start the Development Server:**
  + Run the app locally using Vite:  
    **npm run dev**
  + Open the URL printed in the terminal (usually http://localhost:5173) in your browser.
* **Using the App:**Open the application in a web browser.
  + You will see an 8×8 chessboard with alternating black and white squares.
  + Click on any white square to change its color to yellow.
  + Click on any black square to change its color to red.
  + The changed colors remain until you refresh the page.

1. **ScreenShots**

****

****

1. **Challenges and Solution**

There were no major challenges during the development of this application. Initially, there was an uncertainty about whether to include the backend or not, as the task description didn’t specify it with help of online resources , so I decided to keep the application front-end-only to maintain its simplicity. The main learning part was understanding the Redux toolkit and integrating it with React to manage component state.

**10. Conclusion**

The ChessBoard Game shows how React and Redux Toolkit can be used to build an interactive web application. It displays an 8×8 chessboard and changes the color of squares when clicked. This project helped in understanding how to manage state in a clear and organized way while keeping the app simple and easy to use.

**9. Reference**

* [**React Documentation**](https://react.dev/)
* [**Redux Toolkit Documentation**](https://redux-toolkit.js.org/)
* [**Vite Documentation**](https://vite.dev/guide/)