Advanced Programming Spring 2025

Assignment 3

Release Date: April 19, 2025

Due Date: 10:00 AM, May 03, 2025

Setup Instructions

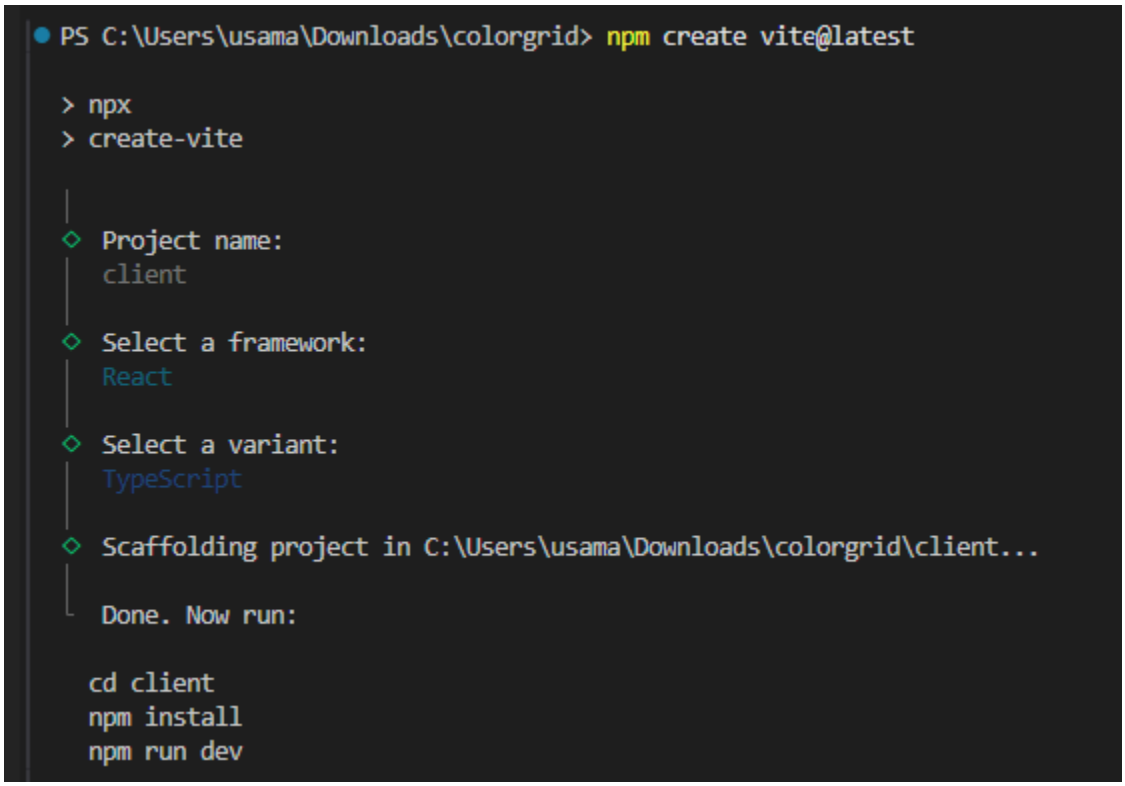
This assignment uses the MERN stack (MongoDB, Express, React, Node.js) along with Socket.IO for real-time communication. A basic project skeleton is already provided. Each folder is currently empty, apart from a base structure to get started.

Since the course is not about styling a web page, a very bare-bones design guide with the `HTML` and `CSS` for the different pages (e.g, login, signup, home, player_waiting, match_found, gameplay, history, profile, leaderboard) have been provided in the `design` folder. You are responsible for splitting them into components for use in your app. These should be enough. However, if you want to add your own styling and/or components, feel free to do so. For those of you more experienced in web dev, feel free to use Tailwind CSS if you want to make any edits.

Note: Any css styling in index.css in the React setup is global and applies across the app, so you may need to change some CSS class names in order to avoid clashes.

Frontend

You can setup your client using [Getting Started | Vite] or see this:



Install Dependencies

Open terminal in both `client/` and `server/` folders and run:

`npm install (you may install any packages if needed e.g. axios, socket.io-client, etc)`

Create `config.env` file inside `server/` for environment variables: (optional)

```
PORT=8000 (for server)
MONGO_URI=<your_mongodb_connection_string>
# or any other variable
```

Start the App

In one terminal window, start the server:

```
cd server
npm run dev
```

In another terminal, start the client:

```
cd client
npm run dev
```

Your app should be running at http://localhost:5173

Helpful Notes:

- Backend uses Express and Socket.IO for real-time game handling.
- Frontend uses React, hooks, components, and socket events to render and sync game state.
- Use the starter HTML/CSS files as reference to convert them into React components.

ColorGrid is a 2-player, turn-based color conquest game inspired by simple logic mechanics like “Tic Tac Toe” and "Connect X".

- Each player joins a match and is assigned a random color.
- A 5x5 grid is displayed.
- On their turn, a player selects a cell which gets filled with their color.
- The game ends when all cells are filled.
- The winner is the player with the larger connected block (island) of their own color.
    - An Island is vertically or horizontally connected cells (no diagonals)
    - You can use **server/utils/maxAreaOfIsland.ts** and compare max areas of both players
- Draws are possible.
- Real-time gameplay is managed using socket events. No page refreshes.

Client Routes

1. `/`
    - Welcome page
    - Buttons: **Login**, **Sign up**

2. `/login`
    - User login form (username, password)
    - On success, redirect to `/home`

3. `/signup`
    - Signup form with username (display name), password, profile picture URL (optional)
    - Each new user starts with **1000 coins**

- On success, redirect to `/home`

4. `/home`
    - Main dashboard
    - Buttons: **Play**, **Leaderboard**, **History**

5. `/newgame/waiting`
    - Starts matchmaking
    - Displays opponent's info once matched or "Waiting for Opponent…"
    - Player should be able to cancel anytime before match is found using **Cancel** button
    - Once match is found, the player should not be able to cancel. You can *briefly* display opponent's info before redirecting to `/newgame/:game_id`
    - Game page is displayed to both players and real-time game begins (see game logic below)

6. `/history`
    - Lists all past games for the logged-in user
    - Format: "`Game #game_id Opponent Result`" e.g.
        i. "`Game 101 Bob Won`"
        ii. "`Game 205 Charlie Lost`"
    - Clicking an entry takes you to `/history/:game_id` to view game snapshot (final state of game)

7. `/update-profile`
    - Update profile form: username, password, profile picture URL
    - Clicking "Save" updates user profile in database

8. `/leaderboard`

- Displays top players sorted by coin score
- Format: “`username won/lost/draw coin_balance`”
  i. “`Charlie 4/3/0 1200`”
  ii. “`Bob 2/3/1 800`”
- Allow searching top players by username

Game Logic (Socket-Based Flow)

1. You can create Socket Rooms: **waiting** (players waiting for a match), **in-game** (2 players max) and socket events for joining these rooms
2. Client emits `find_match` and server adds it to **waiting** players.
3. Two **waiting** players are randomly matched by the server.
4. An **in-game** room is created with a unique `game_id`.
5. Each player is assigned a **different** color (randomly or red-blue).
6. The socket server emits `start_game` to both players.
7. The game begins. A 5x5 grid is shown with a header:
   `Username1 VS Username2 (with profile pics)`
8. Below grid, the **game status** is shown:
   - `Status: Your Turn`
   - `Status: Opponent Turn`
   - `Status: You Won (+200 coins)`
   - `Status: You Lost (-200 coins) or You Lost (if player has 0 coins)`
   - `Status: Draw`
9. When a user clicks a tile on their turn:
   - It changes to the player’s color.
   - Turn is switched to the opponent.
   - Socket emits `move_made` to both players to sync their game states.
10. If player forfeits using **Forfeit** button (below status):

    - Game ends immediately.
    - Opponent wins by default.
11. Game ends when all 25 tiles are filled.
    - Server checks who won/lost or is it a draw and updates the database
    - Server emits `game_end` with the result to both players.
    - Display **Play Again** button at frontend that redirects to `/newgame/waiting` when clicked
    - Update the coins in Navbar

Navigation and UI Hints

- Navbar (persistent) appears once the user is logged in.
    - Left side: Game icon + name → navigates to `/home`
    - Right side: coin balance, profile picture + username dropdown [update profile, logout]

Example Socket Events

Client emits:

- `find_match` → server adds player to list of waiting players
- `make_move` → sends player's game state (grid or cell coordinates)
- `forfeit_game`

Server emits:

- `start_game` → send both player info (e.g. username)
- `move_made` → broadcast move to both players

- `game_end` → broadcast result to both players

Example MongoDB Schema

Users

{ _id, username, password, profile_picture_url (optional), coins }

Games

{ _id, player1_id, player2_id, player1_color, player2_color, final_grid: array[5][5], result, winner_id }

Final Words

You are free to structure your React app as you like, but the following are **required**:

- React functional components
- useState, useEffect, useContext/hooks where applicable
- Socket.IO for all real-time events (no polling)
- Navigation using React Router
- Proper state handling during gameplay
- No page reloads during gameplay
- Show user feedback via game status area
- Basic validations on forms
- Minimal CSS customization (at least bearable); use the design provided
- Do not access the database at frontend
- You are free to implement the backend as you like

As always, **Start Early** and **Happy Coding!**