Project Description

1. End Product Description:

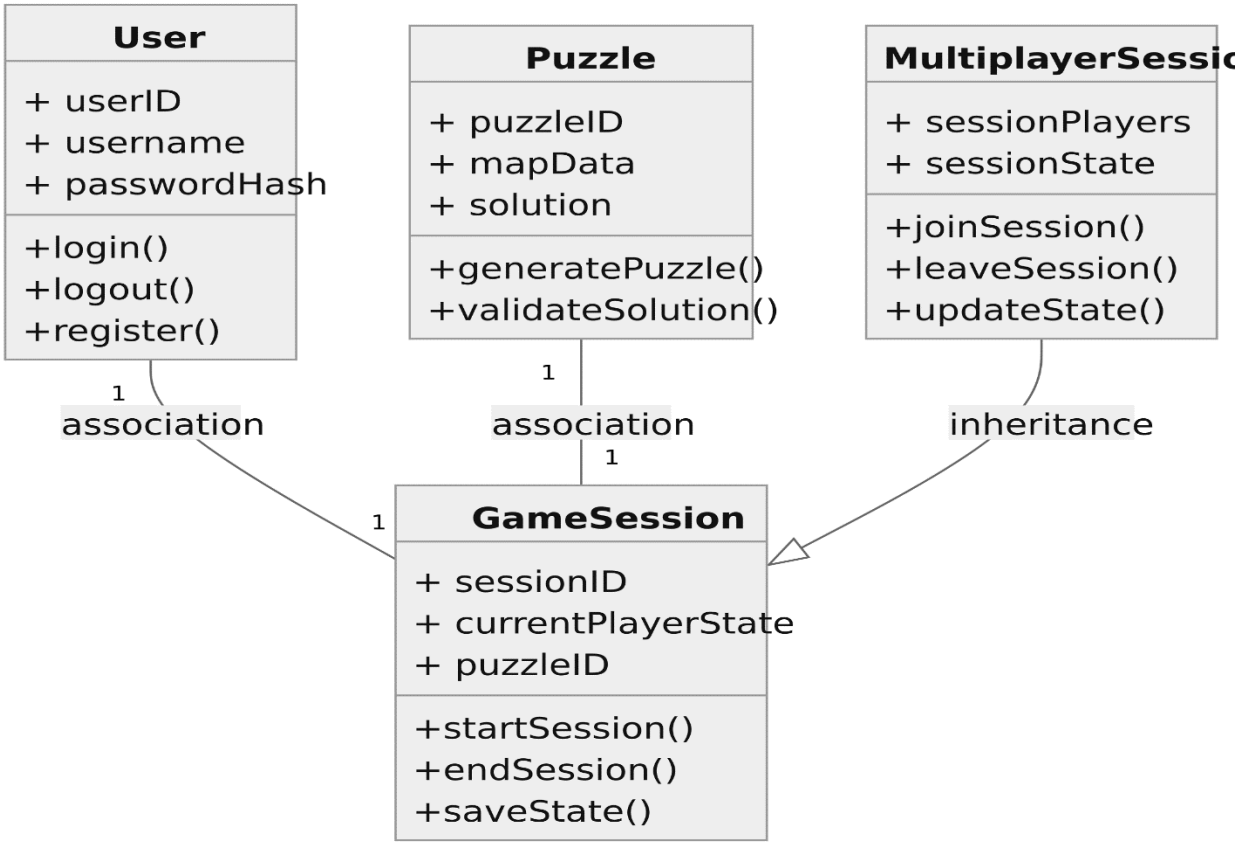
The end product is a web-based Map Coloring Puzzle Solver game, designed to challenge users to color a map with the minimum number of colors without adjacent regions sharing the same color. This application leverages React and Vite for the frontend, with Bootstrap for styling, and Flask coupled with PostgreSQL for backend management. The game supports both solo play and, as an added feature, multiplayer functionality where users can compete or collaborate on puzzles.

2. Flowcharts/UML Notations & ERD:

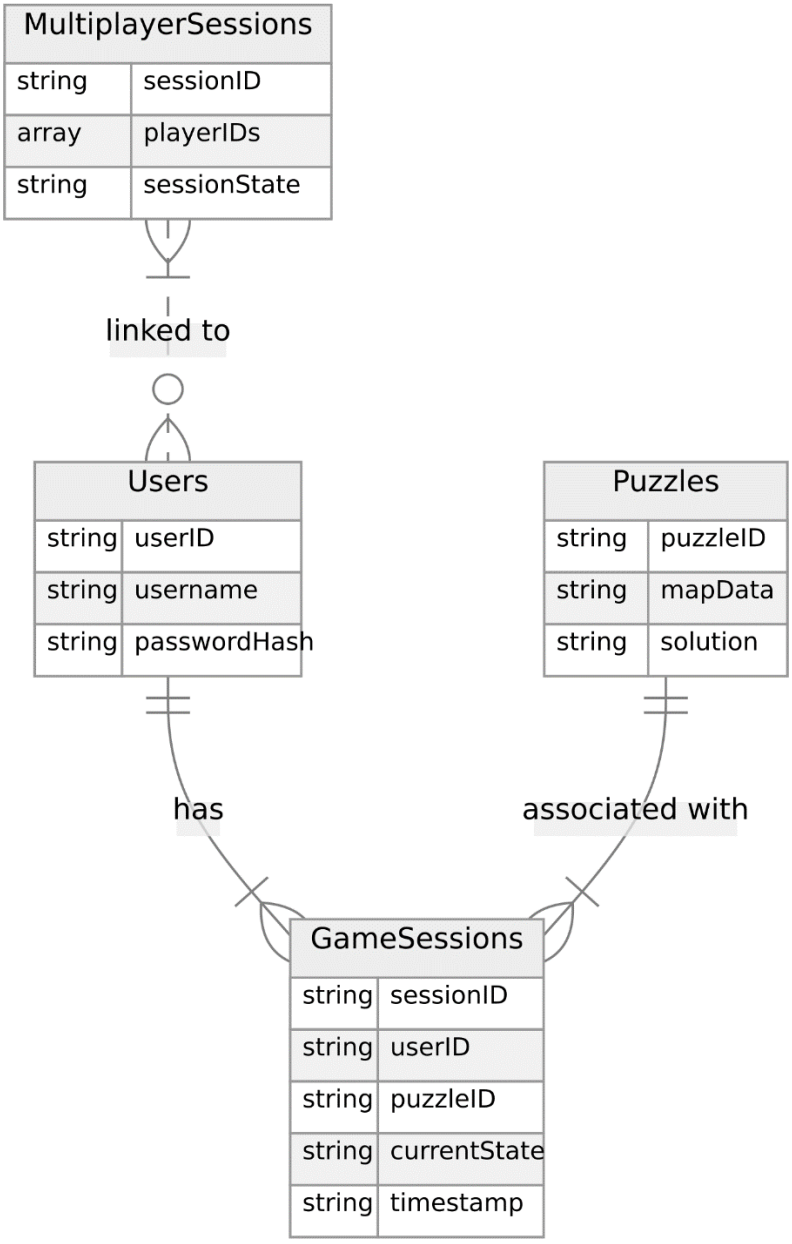
The application architecture follows the MVC (Model-View-Controller) pattern, with React components managing the view, Flask controllers handling API requests, and PostgreSQL serving as the model with tables for users, game states, and multiplayer sessions.

ERD includes tables for Users, GameStates (with columns for user ID, game ID, region colors, and timestamps), and MultiplayerSessions (linking users to shared game instances).

UML Diagram:



ERD:



3. Methods Used in the Puzzle Solver

The core puzzle-solving algorithm employs a backtracking method optimized for the map coloring problem. This algorithm iteratively assigns colors to regions, ensuring no two adjacent regions share the same color, and backtracks when it encounters a conflict, searching for an alternative path.

Pseudo Code:

```
function solveMapColoring(map):
  if all regions are colored:
    return True
  for each color in colors:
    if color is safe for region:
      assign color to region
      if solveMapColoring(map) is True:
        return True
      remove color from region (backtrack)
  return False
```

Market Space and Selling Points

1. Market Space: The application targets educational sectors, puzzle enthusiasts, and individuals interested in algorithms and problem-solving strategies.
2. Selling Points: It offers a unique blend of entertainment and education, enhancing logical thinking and planning skills. The multiplayer feature fosters collaboration and competition, expanding its appeal.

Functional Specifications

Product Features:

1. Interactive Map Interface: Users can click on regions to apply colors, with immediate visual feedback on adjacency conflicts.
2. Solver and Hints: At any point, players can request the solver to finish the puzzle or hints for the next move.
3. Partial Solution: User can get a partial solution to solve the next or remaining coloring.
4. Generate new maps: user can generate new maps just clicking on generate maps button.
5. Reset Button: User can reset the coloring he has done if he gets stuck.
6. Timer: There will be a timer to complete the coloring

Deployment

To deploy the Flask backend to Heroku:

Prepare the Application:

1. Ensure requirements.txt is updated with all dependencies.
2. Include a Procfile with the command to run the Flask app, e.g., web: gunicorn app:app.

Deploy on Heroku:

1. Use the Heroku CLI to log in, create a new app, and link the Git repository.
2. Set up PostgreSQL on Heroku and configure the database URL in the Flask app.
3. Push the code to Heroku using Git, and Heroku will automatically deploy the app.

Environment Variables:

1. Use Heroku's dashboard or CLI to set environment variables for database connections and any other sensitive configurations.

Milestones and Features

1. M1 (Setup and Initial Design): Establish project structure, design database schema, and set up basic Flask and React applications.
2. M2 (Core Functionality): Implement the map coloring logic and basic UI interactions, along with Flask API endpoints for game state management.
3. M3 (Advanced Features): Develop the puzzle solver, hints system, and start on multiplayer functionality.
4. M4 (Testing and Deployment): Conduct thorough testing, apply final polish, and deploy the project to Heroku.