Testing and Databases

Also assignment 14, assignment 15 and Final Project.

Overview

In this program you will create an application to create a shared game of tictactoe. Multiple pairs of people should be able to play at the same time.

GamePlay Overview:

- Initial Page:
    - Shows an input for userName and a box for gameID.
    - To start a totally new game the user will enter their userName and leave the GameID empty. This will cause the system to return a gameID.
    - The first player will give their partner the gameId.
    - The second player will enter their name and the gameId.
- This will commence the game:
- Player1 will be prompted to select a square and will send a move api call.
    - if its invalid player1 will receive an error message
- Player2 wil then be prompted to move
- Play will continue until someone wins or there is a draw (all squares filled)
- At that point each player will be given a score screen showing their and all other players scores as well as an option to play each other again or to play a new game.

This assignment will be done in multiple parts.

- In Assignment 14 you will focus on the datamodel.
- In Assignment 15 you will focus on the API
- In the final project you will focus on the Client Side

Assignment Details:

- Create a new directory called Assignment-14
- In this directory create a directory called model and test. Put your data model code in the directory model (called game.js) and your mocha tests in test directory.
- Make sure this is in git.

For Assignment 14:

You will first create a series of Mocha/Chai tests to get ready for your code. You will put these in git and submit Assignment-13-tests for me to approve. You may start on the actual datamodel code while I am approving the tests but it might take you several iterations to get the tests approved.

Data Model Details:

- gameSchema
    - gameId: String//random 6 letter, lower case string to identify game. You create thi sincode
    - player1Name: String
    - player2Name: String
    - moveCnt: Number
    - lastMoveTime: Number -> localtime, seconds when last move was updated
    - board: [Number] -> array of 9 number representing board
    - 0 1 2
    - 3 4 5
    - 6 7 8
    - state: String // waiting, player1Turn, player2Turn, player1Win, player2Win, stalemate

functions:

- async function createId()
    - function to create a randome game id and make sure it does not exist in the database
- function getGame(gameId)
    - returns a promise to find the game given the game id
- function newGame(playerName, gameId)
    - returns a promise to setup a new game. See writeup for details
- function move(gameId,playerName,move)
    - gameId -> id of valid game
    - playerName -> string name of player
    - move - 0-8 indicating which square player moves to
    - returns a promise to make a move:
        - if ok: returns the gameModel
        - if error calls reject with an error message.
    - example errors: trying to move when not your turn, trying to move when game won…
- function getGames
    - returns promise to return an array of games. Please make sure to delete the gameID from the individual games so people can't hijack other games.

- async function clear -> I added this helper function to clear out the database for testing
- function testAdd(gameSchemaInstance)
    - I added this helper function to directly create a game instances.
    - Used for testing.
    - Returns a promise to add the data.

Step1:

- Create tests using Mocha and Chai to test this api.
- Create a strong suite of tests that will prove the datamodel.
- Create these in test/*.js
- Add these tests to git
- submit this to Assignment-14-tests

Step2:

- Build the data model. It should pass all the tests you created.
- Submit the code to git
- submit a link to canvas Assignment-14-data

For Assignment 15

Copy over Assignment-14 to Assignment-15 in your working copy on on git.

Using express, create the following API's that will invoke the data models created in the previous assignment:

- GetGames
    - url: /api/v1/games
    - method: get
    - json_in: N/A
    - json_out:
        - {games:[game1,game2…]|
        - please make sure that no gameId's are returned, simply return "" for gameId for all games
- play
    - this method is invoked to start a game
    - url: /api/v1/game
    - method: post
    - json_in: {playerName: String, gameID: String}
    - json_out: {status: OK or FAIL, msg=String, game: gameSchema for existing game

- move
    - this method is invoked to make a move
    - url: /api/v1/move/:gameID/:playerName/:movePosition
    - method: get
    - json_in: none
    - json_out: {status: OK or FAIL, msg=String, game: gameSchema for existing game
- More Details
    - Also create a series of SuperTests for testing the api. Make sure to document and include these tests in your git.
    - These tests (and of course the code) should include security tests to make sure your api properly sanitizes and escapes user input.
    - Run this server on port 3015
    - submit a link to the api and to your git repo.

For Final Project

Copy over Assignment-15 to FinalProject
I am providing in my public git the html and javascript files to play the game. I have not heavily tested this code so make sure it works and fix bugs.
https://gitlab.csi.miamioh.edu/campbest/cse270e-campbest-public

There is a "manage boards" link at the bottom of the page.

You are to create a nodejs based page (eg: Not using ajax) management menu page with links to the following: Each are separate pages

1) Statistics page: Create a table that lists each player, how many times have they won and lost and their percentages.
2) Page that lists all games and their details. eg: player1name,player2name, outcome, number moves and the final board. (board should be shown as a tic-tac-toe board, not an array).
3) In Progress list: Create a table that lists all games that are in progress: eg: are not won, lost or stalemated. List game details of names and time of last move.
4) Orphaned game list: create a table that lists all games in the waiting state and the time they were created along with player1name.
5) Page to clear out old games. It should have a form that lists a date and will clear out all games that are older than that date. This page should require a password to clear out the data and the password shall be "CLEAR".

Run the program on port 3016
submit links to the program and to your git repository