

Project report - Stack underflow

Ilija Simić, Ledio Jahaj and Inti Gabriel Mendoza Estrada

1 Introduction

The goal of this project was to implement multiplayer checkers game. This app should support basic user devoted tasks, such as login and registration, provide list of all possible online users and functionality of checkers board.

2 Architecture and Technologies

Communication in the application between server and client is done via HTTP using REST API or via Web Sockets. It will be explained in more detail in following sections. Technologies used for server side are *Node.js* with *Express* framework and *MongoDB* for database. In the client side we did not use any framework, just *vanilla JS* with *toastr* library for animations.

3 Server side

Back end code is organised in *index.js* file and *api* directory. In *index.js* we initialized connection to database, routes, web socket, static content and added port where server should listen for requests.

In directory *api/models* we have described *User* database model and added function for validation of UserDTO(DTO - data transfer object) using *@hapi/joi* library.

In file *routes/routes.js*, we registered *REST API* paths for login(POST method on /login) registration(POST method on /register). In file *routes/sockets.js* we initialize Web Socket on port 9000. For Web Socket we used library *ws*. Socket class inside himself has a map, where keys represent user username and values represent web socket connection. When user connects to the socket, he is added to the map and all other users are notified about new online user. When the user closes the page or logout, the connection is closed, the user is removed from the map and all others are notified about it.

In directory *controllers* we have methods for login and registration, which are called from *routes* when the server receives corresponding request. Both methods firstly check the data from HTTP body and provide HTTP response. These methods also communicate with database.

4 Client side

Client side files are located in *webapp* directory. There are four sub-directories *login*, *registration*, *arena* and *game*, each representing different page.

4.1 Registration page

Registration directory has three files: *index.html*, *script.js* and *style.css*. In the *index.html*, we have just included script and library files and added one *div* element with an id *root*. In *script.js*, we have three classes *Model*, *View* and *Controller*, for the **MVC pattern**. *Model* class contains only constructor, where we initialize empty data about user and status(which would be received from server). *Model* also posses method for sending POST request on the server(**REST API**). In *View* class we have added necessary input fields for the **Web Forms** and added *binding methods for event listeners*. In *Controller* class we have references to the *Model* and *View* class and methods which are bound with *View* methods. These methods are for checking if password and repeat password are equal and for retrieving user data form *View* and sending it to *Model* which submits request. Unsuccessful request is followed by red toastr notification, while successful will be followed by green and redirection to login page after hiding of toastr.

4.2 Login page

Login page has the same structure and very similar behaviour to the registration page. Only difference is that it only has two fields(username and password) and does not have password validation. After the successful login, in the *SessionStorage* we save a token like information - username. The user is then redirected to Arena page.

4.3 Arena page

Arena page has the same file structure as two previous pages. The goal of arena page is to show all online users and provide possibility for starting the game against one of these. It uses **Web Sockets** for retrieving list of all online users. In *script.js* we also have classes *Model*, *View* and *Controller* for the **MVC pattern**. *Model* has a current user(retrieved from *SessionStorage*) and the list of all online users. *View* has dynamic content. Either will render *div* element with message that the user is only online user, or it will render *li* elements with list of online users name and button for playing game against

that user. It also has a logout button, which removes user from the storage and redirects back to the login page. *Controller* connects to socket retrieve list of online users, removes the current user and render corresponding view. It also has opportunity to receive a message for starting new game from other user. This will create a toast, after whose hiding, app redirects to the game page.

4.4 Game page

Game directory has different structure, it has *game.html* and *draw.js* file. In html file, we have drawn red and black rectangles(SVG technology), for the board. It also has download button and area for **Drag and Drop**. In the *draw.js* file as usual, we have **MVC pattern**. *Model* contains matrix, which represent game state. In it zero is empty field, 1 represents white checker(SVG circle), while 2 represents green. In *View* class we have method for setting checkers based on arbitrary matrix. This method is then called from *Controller* class where argument is provided as model matrix. Besides binding methods, controller also contains methods for download game state as json and for handling Drag and Drop events. It prevents Drag Enter and Drag Over result and on Drag Drop reads the file content using *FileReader*. After that we call function for rerendering circle positions.

5 Screenshots

Figure 1: Register page - Web Form custom validation

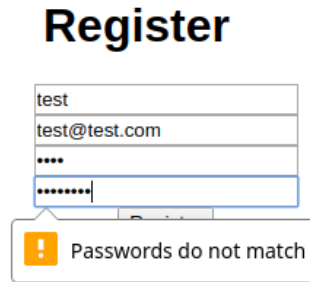


Figure 2: Register page - reject response from the server

Register

❗ User with that username already exists!

Figure 3: Login page - successful login

Login

✔ Welcome ttt

Figure 4: Arena page - only online user

Online players

You are only online player.

ttt(logout)

Figure 5: Arena page - before starting game

Online players

✔ ttt Play

mmm(logout)

Online players

✔ mmm Play

ttt(logout)

Figure 6: Game page - board and checkers

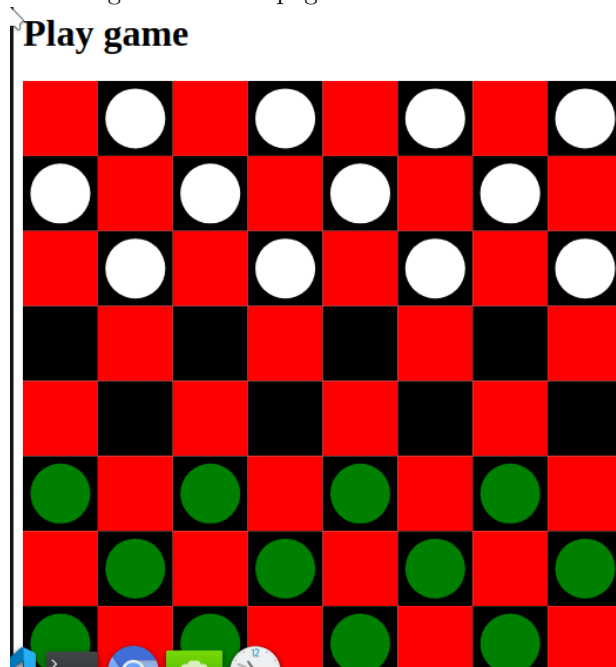
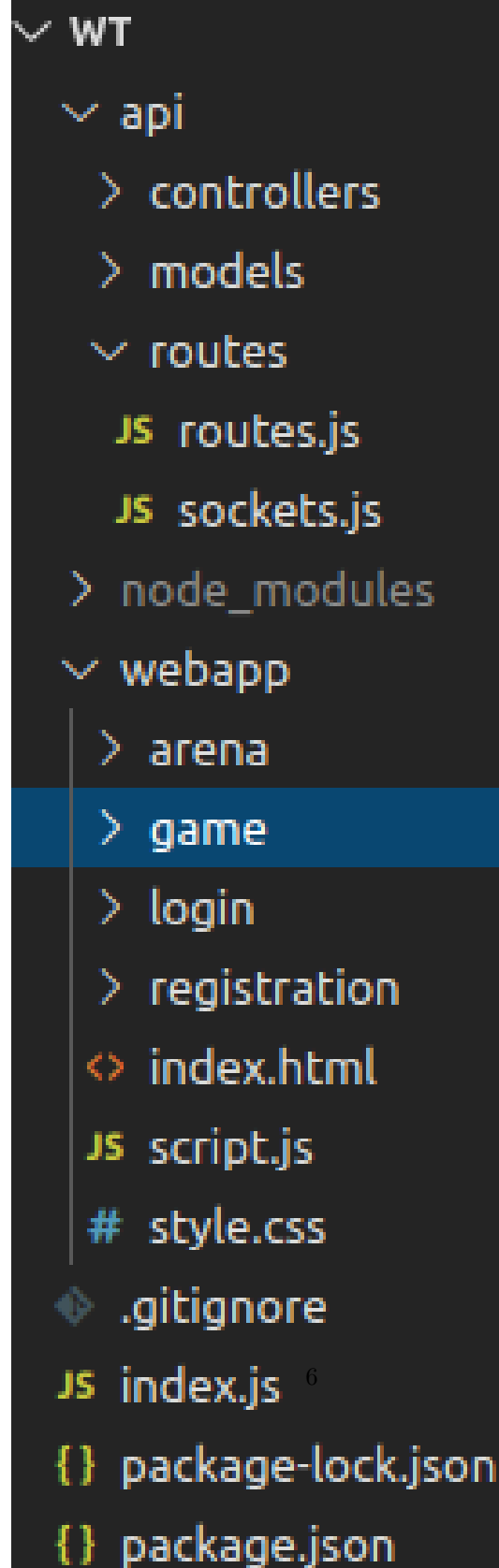


Figure 7: Project organization



The image shows a file explorer interface with a dark background. The file tree is as follows:

- WT
 - api
 - controllers
 - models
 - routes
 - routes.js
 - sockets.js
 - node_modules
 - webapp
 - arena
 - game
 - login
 - registration
 - index.html
 - script.js
 - style.css
 - .gitignore
 - index.js
 - package-lock.json
 - package.json

The 'game' directory under 'webapp' is selected and highlighted with a blue background.