EXAM: Sistemes i Tecnologies Web June 2024

Niu: Nom:

☞ Tic-tac-toe

After submitting via moixero.uab.cat, fill in the following box.

Electronic Submission

The SHA1 checksum of the received file is:

....

Time stamp is:

....

Correction guide:

The grade will be...	When...	Scoring guide
From 0 to 5 points	When the solution does not fully work and there are *serious* conceptual errors in any of the following key elements: callbacks, closures, classes, inheritance, promises, and the various elements of Vue (reactivity, directives, events, props, components, ...).	Starting from 5, 1 point is deducted for each conceptual error.
From 5 to 8 points	When the solution does not fully work but there are no serious conceptual errors.	Starting from 8, 0.25 points are deducted for each error.
10 points	The exam is validated by a professor and submitted it electronically.	You have a 10. You have earned it.

Instructions

Follow the instructions below to boot the lab machine and import the project skeleton.

- Start the machine if you haven't started it already and select the Linux partition (user:exam and passwd:exam).
- Open a console: Applications → Terminal.
- Download the project skeleton by running the command:

```
wget https://moixero.uab.cat/ExamenSTW.7z
```

- Unzip the file.

```
7z x ExamSTW.7z
```

- You have the skeleton in the directory **stw**. Take the exam (you can use the same terminal you already have opened, and open the **stw** directory with Visual Studio Code).
- To run the application:
  1. the client: `npm run dev`
  2. the server: Go to the `src` directory and run `node exam.js` or `nodemon exam.js`.
- Note: If you are using Chrome, you will see that it opens in incognito mode and the Vue addon is not available. Open a new Chrome (Ctrl+N). The new window is no longer incognito and the Vue addon can be used.
- Once you have finished developing the project, you must submit your code electronically. **If the entire application works for you, please let us know before submitting electronically.**

- To deliver electronically, create a zip as follows:
  Move to the `src` directory

```
7z a sol.zip exam.js App.vue components/SellForm.vue components/Shop.vue
```

- Check the contents of the file you will deliver (open the file and look at the contents of the files inside).
- Once you know for sure that you want to deliver this file, upload it to: `https://moixero.uab.cat/`.
- Write down the two values provided by the server (the checksum and the timestamp) on the exam and give it to the supervisor.
- **When you're done, don't log out and don't stop the machine!!!!!!!**

Context

We must implement a simplified version of the tic-tac-toe game. Using two tabs in the browser, each tab will represent a player of the game (Figure 1). To make it easier, each player will see just their movements. There are two use cases where the player will see in her board the other's player's token:

- If it's the player's turn to play and clicks on a cell occupied by the other's player token (Figure 9).
- If it's not the player's turn to play and clicks on a cell occupied by the other's player token (Figure 8).

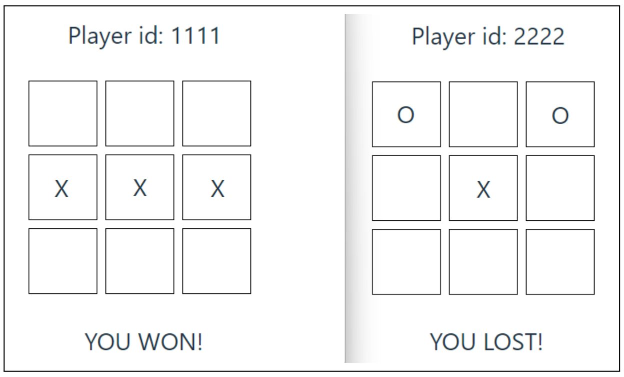

Figure 1: Player1 (P1) plays with 'X' and Player2 (P2) with '0'. P1 wins. Notice that P2 cannot see all the movements of P1 and vice versa.

You can start this portal by running:

1. The server: Go to the **`src`** directory and run **`node exam.js`** or **`nodemon exam.js`**.
   The server will be available through the URL `http://localhost:3001`.
2. The client: **`npm run dev`**
   The client will be available through the URL `http://localhost:3000`.

You will see the view in figure 2.

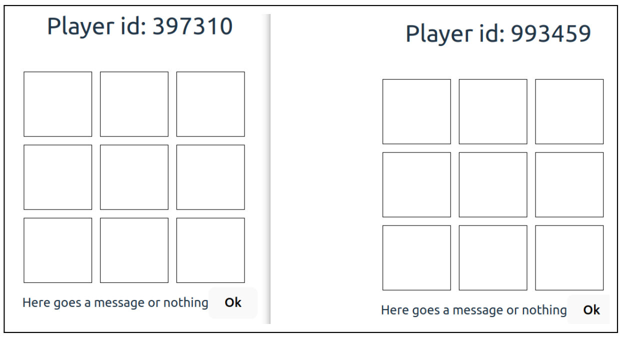
Player id: 397310
Here goes a message or nothing
Ok
Player id: 993459
Here goes a message or nothing
Ok

Figure 2: Skeleton view.

Frontend exercise

You need to complete the **`RootComponent`** (**`App.vue`**) and the **`Cell`** component. Below are the component specifications.

RootComponent (App.vue)

The **`RootComponent (App.vue)`**, will create a 3x3 grid of **`Cell`** components (**done**).

Specifications

- Receives the **`gameWinner`** from the **`Cell`** component through a **`v-model`** binding. If the **`gameWinner`** is *truthy* and corresponds to the **`RootComponent's playerId`** variable, it shows the message **`YOU WON!`**, otherwise shows the message **`YOU LOST!`** (Figure 1)
- Reacts to the event **`message`** trigered by the **`Cell`** component and stores the event parameter in the reactive variable **`message`**. It shows the message followed by an "OK" button (Figure 7).
- When the message's **`OK`** button is clicked, it sets the **`message`** to undefined so neither the message nor the button are longer visible.

Cell component

The **`Cell`** component will be the one to manage cell clicking, the communication with the backend, and displaying the token (`'X'` or `'O'`).

Specifications

- It will receive the following **props**:
  - **row** (*number*): Represents the row of the cell. Goes from 0 (the upmost row) to 2 (the downmost row).
  - **column** (*number*): Represents the column of the cell. Goes from 0 (the leftmost column) to 2 (the rightmost column).
  - **playerId** (*number*): Represents the id of our player.
  - **gameWinner** (*number*): Represents the id of the player who has won the game. Defaults to **`null`**. It is two way bound with the **`RootComponent`**.
    Remember that the following two tags are equivalent:
    ```
    <Component v-model:specificProp="data" />
    <Component :specificProp="data" @update:specificProp="x => data = x" />
    ```
- In the **template**, the tag **`div class="cellExterior"`** represents the whole cell. We must listen to its **`click`** event. Once clicked, we must perform a request to the endpoint **`/cell\_click`**:
  ```
  fetch(`http://localhost:3001/cell_click?playerId=11111&row=0&column=0`)
            .then(res => {this.status=res.status; return res.json()})
            .then(json => { ... })
            .catch(() => { ... })
  ```
  ☞ Clicking on a cell will have no effect if we are already performing a request, or there is already a winner or there is a token in the cell.
- The endpoint replys with a JSON and an status code. The JSON can be:
  - **`{gameWinner: null, token: X}`** where token can be '**`X`**' or '**`O`**'. There is still no winner. In this case we just show the received token in the cell. To do so, in the **template**, the **`<p>`** tag shows the token. The token value by defaul is an empty string `""`. It will change to `"..."` while waiting a response to the request to the **`/cell_click`** endpoint. After receiving the response, it will change to the received token value '**`O`**' or '**`X`**' (Figure 3).

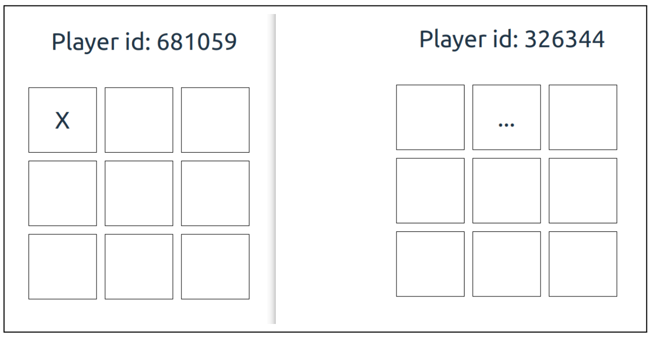

Figure 3: P1 has clicked on the cell[0][0] and got the '**`X`**' token. P2 clicked on c[0][1] and is waiting for the server response.

– **`{gameWinner: 11111, token: X}`** In this case the game has finished. We have a winner, player 11111 which has performed the last movement. We need to notify the **`RootComponent`** about the winner so it can display the message "YOU WON" (Figure 4).

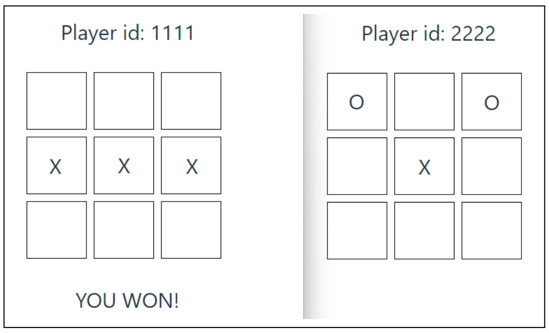

Figure 4: P1 wins, P2 still doesn't know because it is her turn to play.

– **`{gameWinner: 11111}`** Player 1111 won the game in the last movement. Player 2222 still doesn't know as it is her turn to play. Player 2222 clicks on a cell and gets this JSON. We need to notify the **`RootComponent`** about the winner so it can display the message "YOU LOST" (Figure 5).

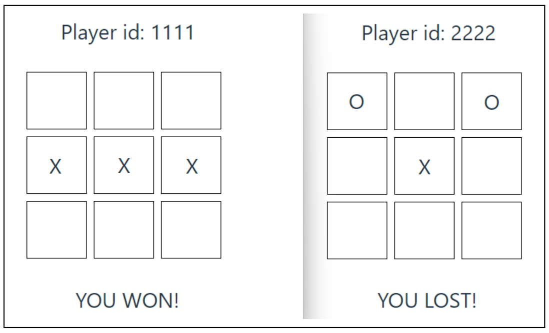

Figure 5: P1 wins, P2 clicks on a cell and gets from the server that the game is over and P1 is the winner.

– The status code (**`this.status`**) can be:

200: - It was our turn and the cell was empty. In this case we show the token received from the server (Figure 6).

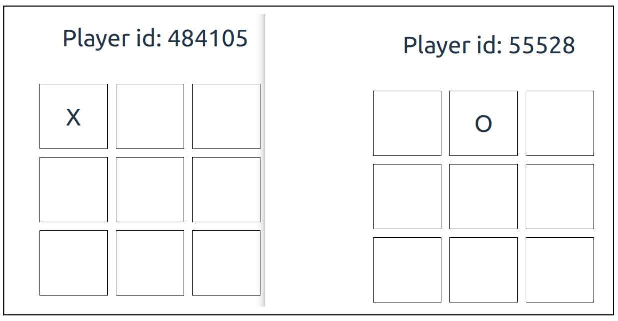

Figure 6: P1 selects c[0][0] and P2 c[0][1].

- You won the game (Figure 4) or you lost the game (Figure 5).

401: It was not our turn (Figure 7). We trigger the event **message** passing as a parameter the string "It is not your turn" so the **`RootComponent`** can display it.

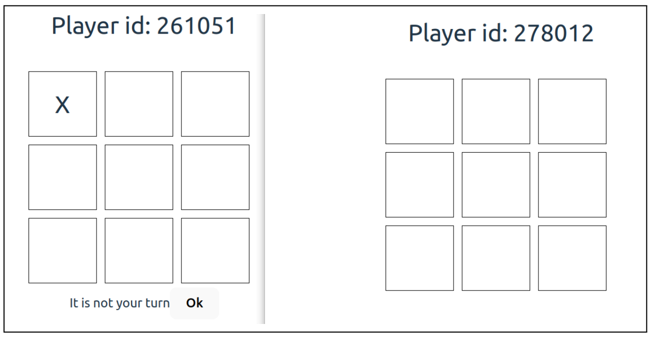

Figure 7: P1 selects c[0][0] and before P2 plays, P1 selects c[0][1]. Notice the message at the bottom of P1's view.

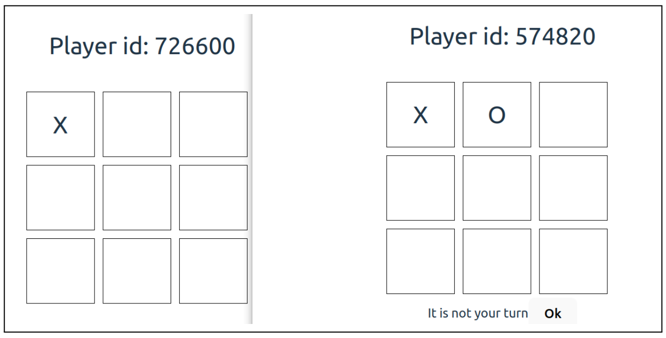

Figure 8: P1 selects c[0][0], P2 selectsc[0][1] and before P1 plays, P2 selects c[0][0]. Notice the P1's token is shown in P2's board.

400: The cell was already set by the other player (Figure 9). In this case, we will show a red border around the cell: `style="border: 2px solid red"` and the received token in the cell.

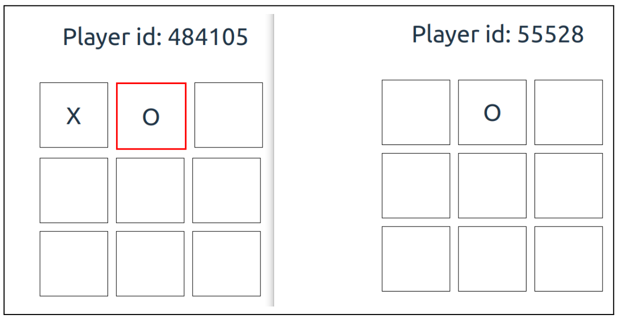

Figure 9: It's P1's turn and selects c[0][1] where there is already a P2 token. Notice that we show P2's token in P1's board.

Testing the front without the backend implementation

We provide you with a dummy server that after one second of receiving the request to the endpoint `/cell_click`, always returns an status code of **200**, and the following JSON: `{gameWinner: null, token: "X"}`
You can change the values of the JSON fields and the status code to test your interface for the following cases:

- **Normal move**: You don't need to touch the JSON. Clicking twice the same cell won't trigger a new request.
- **Cell not empty**: Change the status returned by the server to `400`.
- **Not your turn**: Change the status returned by the server to `401`.
- **You win**: Change the status returned by the server to `200`. Modify the JSON and set the `gameWinner` field with your `playerId`. Since the game is over, you won't be able to click any cell.
- **You lose**: Modify the JSON and set the the `gameWinner` field with the id: 111. Click a cell in the second tab of the browser.

Backend exercise

The overall aim of the backend is to keep track of turns, the board clicked cells, and determine the winner of the game. It is almost **done**. You need to complete the declaration of the module **`game`**, which will offer methods for the endpoint `/cell_click` to use. In the file **exam.js**, only the parts marked as *// TODO* need to be implemented.

game module specifications

– This module will be implemented using the *module pattern*. Here is an example:

```
const testModule = (()=>{
    let _a = 1;
    const increase = () => { _a++; },
    const getValue = () => { return _a; }
    return {
        increase,
        getValue,
    };
})();
```

– (**done**) It defines the following private variables:

- **`currentTurnToken`** (*string*): This variable can be either `"X"` or `"O"`. It indicates the token to be placed on the next cell that will be clicked. Defaults to `"X"`, meaning that the first player that clicks a cell, will be assigned crosses.
- **`tokenOwner`** (*dictionary*): This variable stores the association between a token and the player that owns it. For example, if it contains `{"X": 1111, "O": null}`, means that the crosses are owned by player with id 1111, and that the circles are not owned by anyone yet. It defaults to `{"X": null, "O": null}`, given at the beginning of the game, no player has a token assigned.
- **`board`** (*array*): It is a 3x3 matrix that represents the board of the game. Each position of the matrix (`board[row][col]`) will contain `null`, if the cell has not been clicked yet, or the token (`"X"` or `"O"`).

– It defines the following methods:

- **`isMyTurn(playerId): boolean`** **(already implemented)**: Given a playerId returns true of false wether it's the player turns or not.
- **`getCellToken(row, column): 'X'/'O'`** **(already implemented)**: Returns the token in a cell (`'X'/'O'`) or null if there is no token.
- **`getGameWinner(): playerId`** **(already implemented)**: This method will return the **`gameWinner`** Id or `null` if there is still no winner.
- **`endTurn(): void`** **(already implemented)**: **This method needs to be called once the player clicks on a cell and the cell is empty.** It will swap the `currentTurnToken` value (`'X'/'O'`) and will check if there is a three-in-a-row line on the board.
- **`refreshTokenOwner(playerId): promise`** **(TODO)**: This method receives the **`playerId`** of the player performing the request to `/cell_click` and **returns a promise**.
  The method checks if it is you turn (use method `isMyTurn(playerId)`), if so, updates the token owner (`tokenOwner[currentTurnToken] = playerId`) and **resolves to nothing** the promise. Otherwise it **rejects the promise also to nothing**.
- **`refreshCell(row, col): promise`** **(TODO)**: This method receives the, **`row`** and **`column`** of the clicked cell in the board and returns a promise. **The method checks if a cell is empty**. If the cell is empty (`!board[row][column]`), it places the current player token in the cell:
  `board[row][column] = currentTurnToken`
  and **resolves to the token in the cell** (`currentTurnToken`). If the cell is not empty **rejects to the token in the cell** (`board[row][column]`).

– The **`game`** module needs to reveal (export) the methods: **`refreshTokenOwner`**, **`refreshCell`**, **`getCellToken`**, **`getGameWinner`** and **`endTurn`**.

Specifications of the web server

The server will implement one endpoint: `/cell_click`, which will receive the playerId, and the row and column of the clicked cell:
`http://localhost:3001/cell_click?playerId=11111&row=0&column=0`
We must convert all three parameters from strings to numbers before passing them as parameters to the methods of the module **game**. We can convert them to numbers by using Javascript function `parseInt` (i.e. `parseInt('123')`).

Here are the steps that the `/cell_click` endpoint must perform:

1. Call the function `getGameWinner` to check if the game already has a winner.
2. If there is already a winner send the client the JSON `{gameWinner: thegame_winner}` and status `200`.
   `res.status(200).json({gameWinner: the_gameWinner})`
3. Else, call `refreshTokenOwner` to check it is your play turn.
4. If it is not your turn send the status code `401` to the client. Use `getCellToken` function to get the token in the cell or null:
   `res.status(401).json({gameWinner: the_gameWinner/null, token: 'X'/'O'/null})`
5. If it is your turn, check if the selected cell is empty (`refreshCell`).
6. If the cell is empty the status code will be `200`, if not `400`.
7. Either if the cell is empty or not, get the token the `refreshCell` promise has resolved/rejected to.
8. In case the cell is empty you need to end your turn by calling the function `endTurn()`.
9. Reply to the client with the json:
   `{gameWinner: the_gameWinner/null, token: theCurrentToken}` and the current status (`200`, `400`, or `401`):
   `res.status(status).json({gameWinner: the_gameWinner/null, token: theCurrentToken})`

Application Test

☞ To start a new game you need to restart the server and reload the browser tabs.

1. **Normal round**: P1 to c[0][0] and P2 to c[0][1]. **Output:** Figures 3 and 6.
2. **Click twice over the same cell**: Test_1 plus P1 to c[0][0]. **Output:** No request is generated.
3. **Cell is not empty**: Test_1 plus P1 to c[0][1]. **Output:** Figure 9.
4. **It is not your turn and you click over an empty cell**: Test_3 plus P2 to c[0][2]. **Output:** Figure 7. After clicking the `OK` button, the “It is not your turn” message and the button disappear.
5. **It is not your turn and you click over a not empty cell**: Test_3 plus P2 to c[0][0]. **Output:** Figure 8. After clicking the `OK` button, the “It is not your turn” message and the button disappear.
6. **`P1 wins`**: Go on through the game and make P1 the winner. **Output:** Figure 4. P2 clicks on an empty cell. **Output:** Figure 5.

This is a draft page. Don't unstaple it!