


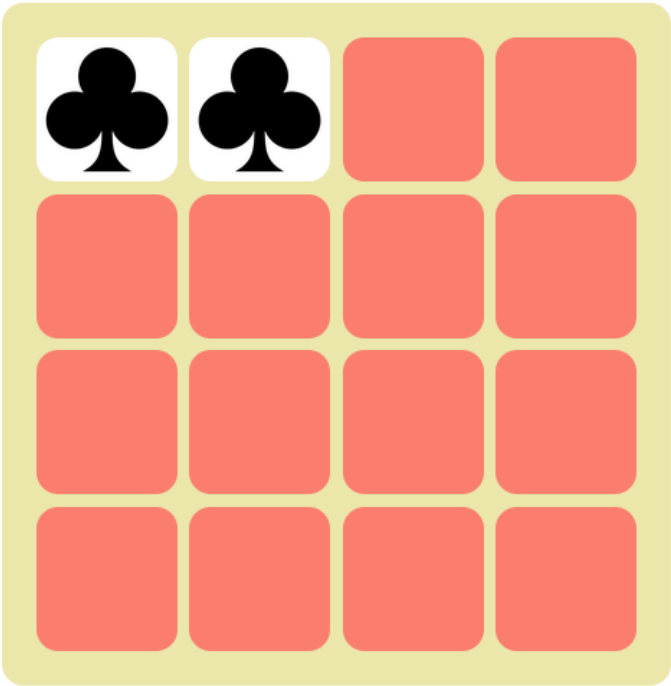
| | | |
|---------------|--------------------|---|
| Subject | CCAPDEV |  <h2>Mini Challenge 3</h2> |
| Deliverable | Mini Challenge 2 | |
| Grouping | Individual | |
| Period | In-Class | |
| Primary Focus | NodeJS and MongoDB | |

Image taken from <https://publicdomainvectors.org/en/free-clipart/Playing-cards-poker-icon/89046.html>



Board code

First Move

Results
 Cell 3 contains C
 Cell 4 contains D
 Cell 3 and 4 do not match!

All images are taken from [game icons.net](https://gameicons.net).

Template

<https://drive.google.com/file/d/19X-xPdCtB0QpF4uUdDjyzIPz5dd59ZE8/view?usp=sharing>

Game Specifications

The game is a kind of memory game. The objective of the game is to select pairs of corresponding cards.

- There are 8 pairs of cards in each game.
- The board is composed of cells where each space has an associated cell number. The cell numbers are as follows:

| | | | | | | | | | |
|--|---|----|---|----|---|----|---|----|---|
| | | | | | | | | | |
| | + | -- | + | -- | + | -- | + | -- | + |
| | | 01 | | 02 | | 03 | | 04 | |
| | + | -- | + | -- | + | -- | + | -- | + |
| | | 05 | | 06 | | 07 | | 08 | |
| | + | -- | + | -- | + | -- | + | -- | + |
| | | 09 | | 10 | | 11 | | 12 | |
| | + | -- | + | -- | + | -- | + | -- | + |
| | | 13 | | 14 | | 15 | | 16 | |
| | + | -- | + | -- | + | -- | + | -- | + |

- For the `board_cards` schema, an array is used. this will accept a list of strings. Creating an entry will look like the following:

```
const array = ["1", "2", "3"];
const instance = Model({
  map: array
});
```

- For this project, Mongoose or MongoDB drivers can be used. The connections are accessible through the variable `app_data`.

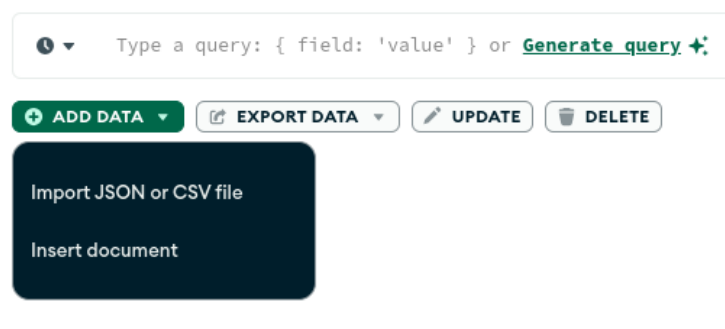
Deliverable Specifications

Only 3 files need to be modified which are **move.js**, **reload.js** and **cerate_new.js**. Each specifies a specific route and the behaviors associated. The file `app.js` may be modified as well but is optional.

- The **create_new** route is used to create a board. A board is a 16 string array where there are pairs of symbols (A, B, C, D, E, F, G, H) arranged in a random order. The function should end by redirecting the user to the board that was just created.
- The **move** route is used to record an action taken by the user and will return the appropriate image. This should take into account the board being used. Each action should be saved in the `select_action` model using `saModel`. The route sends a response which specifies the status of the selected actions and returns the symbol of the two selected cells as well as the image of the first selected cell.
 - The status returned will be labeled “match” when the two cells have the same symbol on the board. If the two are not the same, then the status will be labeled “mismatch”.
 - The values for `location1` and `location2` should be numbers that indicate which cell the data is associated with.
- The **reload** route is used to load all the actions made on the specified board and send it to the client. The actions should be in the same format at the move route's response but will be in the form of an array.

Notes and Hints

- It is suggested to tackle the problems in this order: **create_new > move > reload**.
- To load the json files, first run the application to generate the collections even if there are no entries yet. The using Mongo Compass, load the files:
 - Load **memory.board_cards.json** into the **board_cards** collection.
 - Load **memory.cards.json** into the **cards** collection.
 - To load the data, go to the collection and select the option “Import JSON or CSV file”.



- For `create_new`, the kind of randomization of the board is not specified. Freedom is given as to how this is randomized. A random number generator code is provided.
- For `move` and `reload`, the image is only used when the status returned is a match. If a value is sent when the status is a mismatch, the value will be ignored by the system.
- There is no need to modify `app.js`. However, some solutions can cache certain information to avoid extra database fetches. In those cases, some processing can be done in `app.js`.

Code Notes

- The ID of the saved instance can be obtained like so:

```
//Mongoose
let result = await instance.save();
console.log(String(result['_id']));

//Mongo
let data = { map: map };
let result = await col.insertOne(data);
console.log(String(data._id));
```

- Use async and await for the java asynchronicity.
- It is not advised to fetch each image one at a time.

Submission

- Only submit 4 files, move.js, reload.js, create_new.js and app.js.
- All files are to be submitted into canvas. **Do not zip the files**, simply drop the 4 files as a submission.

Rubric

| Criteria | Description | Percent |
|---------------------|---|---------|
| Submission followed | <ul style="list-style-type: none">The submission instructions must be followed.Submission is late: -10Incorrect file name: -5Missing a file submitted (including app.js): -5 | 10 |
| Create_new Route | <ul style="list-style-type: none">The board must be created and saved.Not saving the board: -20Not creating a board with randomness: -10Not returning proper values: -5 | 35 |
| Move Route | <ul style="list-style-type: none">The action must be stored and the image returned.Action not saved: -10Image not returned: -10Not matching the actions with the map: -15 | 35 |
| Reload Route | <ul style="list-style-type: none">The list of actions must be returned.Not obtaining the list of actions: -10Image not returned: -5Not matching the actions with the map: -5 | 20 |
| Other | <ul style="list-style-type: none">If the code does not compile, the maximum allowed points will be 70.Hard coding any part such as the obtaining of the card images: -20 | 0 |

Note: Additional deductions may be added should a specific case was not included that conflicts with the supposed design direction of the page.