Full Stack Cat VIII

Saving Score and Times Played to Database

* In order to save the Score and the Times Played in the Database, we will need a new model to store that information, and the user_id (the original User _id from Mongodb) and the cat_id (the original CatFancier _id from Mongodb). We will use these other ids much as we did with CatFancier, to call the other databases when we need to re-render the page.

````
const mongoose = require('mongoose');


const GamePlayerSchema = new mongoose.Schema({
    cat_id: {
        type: String,
        required: true,
        trim: true
    },
    user_id: {
        type: String,
        required: true,
        trim: true
    },
    score: {
        type: Number,
        required: true,
        default: 0,
        trim: true
    },
    times_played: {
        type: Number,
        required: true,
        default: 0,
        trim: true
    },

});

module.exports = mongoose.model('GamePlayer', GamePlayerSchema);

````

* For GamePlayer to be of use to us, any time the user hits the /restOfSite route the site will need to be rendered with the information from CatFancier as well as the information from GamePlayer. Each of these routes into '/restOfSite' will have to check to see if the user has played the game yet. If not, the app must store the user info to the database. There are three routes into '/restOfSite', router.get'/auth/dashboard', router.get'/restOfSite/:id' and router.post'/restOfSite' . Here is the code for the router.get'/restOfSite/:id' route, the extra logic can be easily copied and pasted into the other routes:

````
router.get('/:id', ensureAuthenticated, (req, res) => {
   let id = req.params.id;
   CatFancier.findById({ _id: id }, (err, data) => {
      if (err) {
         console.log(err);
      } else {
         let name = data.name;
         let age = data.age;
         let id = data._id;
         let fci = data.favoriteCatImg;
         let user_id = data.user_id;
         GamePlayer.findOne({ cat_id : id }, (err, data) => {
                  if (err) {
                     console.log(err);
                  } else {
                     if (data) {
                        let player_id = data._id;
                        let score = data.score;
                        let times_played = data.times_played;
                        res.render('layouts/site/rest-of-site.ejs', { name, id, age, fci, user_id,
player_id,                                    score, times_played });
                     } else if (!data) {
                        let gamePlayer = new GamePlayer({
                           cat_id : id,
                           user_id : user_id
                        });
                        gamePlayer.save((err, data) => {
                           if (err) {
                              console.log(err);
                           } else {
                              let player_id = data._id;
                              let score = data.score;
                              let times_played = data.times_played;
                              res.render('layouts/site/rest-of-site.ejs', { name, id, age, fci, user_id,
```

```
player_id, score, times_played });
                                    }
                                });
                            }
                        }
                    });
        }
    });
});


````
```

* Open the app and navigate to the game, then check to see if GamePlayer has the user_id etc. in the database (score and timesplayed should still be the default).

### Get the Score to Render From the Database

* The first challenge is to get the score to the javascript.
* The values from the database are being passed to rest-of-site.ejs on the render:

````
```
<div class="profile-bar-div">
<%- include('./partials/profile-bar.ejs') %>
</div>
<div class="portal-container site-container" style="background-image:url(<%= fci %>);">
<div class="portal-content-wrapper game-content-wrapper">
<div class="portal-content game-content">
<div hidden id="name" data-value="<%= name %>"></div>
<div hidden id="age" data-value="<%= age %>"></div>
<div hidden id="fci" data-value="<%= fci %>"></div>
<div hidden id="id" data-value="<%= id %>"></div>
<div hidden id="score" data-value="<%= score %>"></div>
<div hidden id="player_id" data-value="<%= player_id %>"></div>
<div hidden id="times_played" data-value="<%= times_played %>"></div>

<div id="gameDiv" class="game-div"></div>
</div>
</div>
</div>

````
```
```

* In app.js grab these values:

````
const scoreDBDiv = document.getElementById("score");
const score = scoreDBDiv.getAttribute('data-value');
const playerIdDiv = document.getElementById("player_id");
const playerId = playerIdDiv.getAttribute('data-value');
const timesPlayedDiv = document.getElementById("times_played");
const timesPlayed = timesPlayedDiv.getAttribute('data-value');
const nameDiv = document.getElementById("name");
const name = nameDiv.getAttribute('data-value');
console.log(name);

````

* Remove the let score = 0 definition in the javascript variables (because score is now defined above as the data-value passed into the ejs).

````
let randomNumber = 0;
let wins = 0;
let losses = 0;
let val1 = 0;
let val2 = 0;
let val3 = 0;

````

* Score should still render as '0'.

Getting the score to update

* To get the score to update we need to use axios, because we can use it to call routes from the client-side. To use axios in javascript you need to get the library from https://cdnjs.cloudflare.com/ajax/libs/axios/0.20.0-0/axios.min.js or the shared drive (it's on the shared drive.) Make a file in your javascript folder called axios.js and copy the library into it. Hook up the script on top of app.js in page.ejs and defer the script, and you are good to go. It is worth noting that this is the same process if one has access to the internet or not. (Although, with the internet you could just link a cdn, but that can always go away, leaving your app unable to use axios - like this your app is self-sufficient).
* Axios can be used to hit our express mongoose routes and to pass information to them. What axios cannot do when it is being used as we are using it, on the client side, is return information from the database. (This is not always the case, if it is used to call an API it will return that data

to the client, but we are using it to hit a router/route.) Therefore, all our data will update on the client side and that update will be passed to the database.
* What axios will do is return a promise, and in that promise we can render any changed data to the UI, so if the score changes on the UI we can send that change to the database and then use axios to render that change in the promise as if it came from the database (but it didn't).
* In app.js make a function that takes in the changed score and updates it. Have it return the text of the updated score. Run it in matchScores() .

````
function matchScores() {
    if (losses === 3) {
        score--;
        updateScore(score);
        wins = 0;
        losses = 0;
        winsTextP.innerText = "Wins: " + wins;
        lossesTextP.innerText = "Losses: " + losses;
    } else if (wins === 3) {
        score++;
        updateScore(score);
        wins = 0;
        losses = 0;
        winsTextP.innerText = "Wins: " + wins;
        lossesTextP.innerText = "Losses: " + losses;
    }
}

function updateScore(score) {
    let newScore = score;
    return axios.post('/restOfSite/game/game/updateScore/' + playerId, {
        score: newScore
    })
    .then((data) => {
        if (data.status === 200){
            return scoreDBDiv.innerText = "Score: " + score;
        } else {
            console.log("there has been a problem");
        }
    })
    .catch((err) => console.log(err));
}

````

%%%
* It should be noted here that I changed the dynamic html to normal html in rest-of-site.ejs.
* Later, this would prove unnecessary, but as it was already done I did not change it back.
* rest-of-site.ejs now looks like this:

````
<div class="profile-bar-div">
  <%- include('./partials/profile-bar.ejs') %>
</div>
<div class="portal-container site-container" style="background-image:url(<%= fci %>);">
  <div class="portal-content-wrapper game-content-wrapper">
    <div class="portal-content game-content">
      <div hidden id="age" data-value="<%= age %>"></div>
      <div hidden id="id" data-value="<%= id %>"></div>
      <div hidden id="player_id" data-value="<%= player_id %>"></div>
      <div hidden id="times_played" data-value="<%= times_played %>"></div>
      <div id="gameDiv" class="game-div">
        <div id="scoreWrapper" class="score-wrapper">
          <div id="scoreDiv class=score-div">
            <p id="score" data-value="<%= score %>"></p>
            <button id="resetBtn" class="btn reset-btn">Reset</button>
          </div>
          <div id="winsDiv" class="wins-and-losses-div wins-div">
            <p id="winsText" class="winsText"></p>
            <div class="wins-losses-img-div wins-img-div"></div>
          </div>
          <div id="lossesDiv" class="wins-and-losses-div losses-div">
            <p id="lossesText" class="lossesText"></p>
            <div class="wins-losses-img-div losses-img-div"></div>
          </div>
        </div>
        <div id="targetWrapper" class="target-wrapper">
          <div id="targetNumber" class="target-number"></div>
          <div id="playerNumber" class="player-number"></div>
        </div>
        <div id="numberWrapper" class="number-wrapper">
          <button id="target1" class="game-btn target1"></button>
          <button id="target2" class="game-btn target2"></button>
          <button id="target3" class="game-btn target3"></button>
        </div>
      </div>
    </div>
  </div>
````

````
* If you would like to keep the code dynamic, render the reset button as part of the innerHTML of scoreDiv. Something like scoreDiv.innerHTML = "<p>Score: " + score + "</p><button id='resetBtn' class='btn reset-btn'></button>".
The event listener for this button needs to be at the end of setUpGame() or it will not be defined yet. scoreDiv will need to be re-rendered in the axios promise in updateScore().
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

* Next in routes/restOfSite/game make the '/updateScore/:id' route to receive the post request.

````
router.post('/updateScore/:id', (req, res) => {
    let playerId = req.params.id;
    let newScore = req.body.score;
    GamePlayer.findByIdAndUpdate({ _id : playerId }, { $set:{ score : newScore } }, { new: true }, (err,data) => {
        if (err) {
            console.log(err);
        } else {
            let cat_id = data.cat_id;
            res.redirect('/restOfSite/game/game/' + cat_id);
        }
    });
});

````
* You might notice that '/updateScore/:id' does not render anything. Since we already have a GET route, '/restOfSite/:id' that renders the site, we can just redirect to that route.
* Change all the routes into 'rest-of-site.ejs to redirect to the 'router.get'/restOfSite/:id' route. The id is the CatFancier _id. Start with the '/auth/login/dashboard' route:

````
router.get('/dashboard', ensureAuthenticated, async (req, res) => {
    userId = req.user.id
    await User.findById({
        _id: userId
    }, (err, data) => {
        if (err) {
            console.log(err);
        } else {
            let userId = data._id;

```
        // let email = data.email;
         CatFancier.findOne({ user_id : userId }, (err, data) => {
            if (err) {
                console.log(err);
            } else {
                if (!data || data === undefined){
                    res.redirect(`/profile/landing/${userId}`);
                } else {
                    let user_id = data.user_id;
                    let id = data._id;
                    GamePlayer.findOne({ cat_id : id }, (err, data) => {
                        if (err) {
                            console.log(err);
                        } else {
                            if (data) {
                                cat_id = data.cat_id;
                                res.redirect('/restOfSite/game/game/' + cat_id);
                            } else if (!data) {
                                let gamePlayer = new GamePlayer({
                                    cat_id : id,
                                    user_id : user_id
                                });
                                gamePlayer.save((err, data) => {
                                    if (err) {
                                        console.log(err);
                                    } else {
                                        let cat_id = data.cat_id;
                                        res.redirect('/restOfSite/game/game/' + cat_id);
                                    }
                                });
                            }
                        }
                    });
                }
            }
        });
    }
});

});

````
* The 'updateScore' route shown above already redirects to the 'GET' route.

* In the routes there should now only be two ways into 'rest-of-site.ejs' : The GET route and the POST route in routes/site/index.js.
* Move these routes to routes/site/game.js. Fix the redirect paths in routes/auth/login.js router.get '/dashboard' and in ther 'updateScore' route also in routes/restOfSite/game.js.
IMPORTANT: Do not forget to reset the hrefs in the modal close (X) buttons: href="/restOfSite/game/game/<%= id %>">✕ .

````
const router = require('express').Router();
const CatFancier = require('../../models/CatFancier');
const GamePlayer = require('../../models/GamePlayer');
const {
    ensureAuthenticated
} = require('../../passport/auth');


router.get('/game/:id', ensureAuthenticated, (req, res) => {
    let id = req.params.id;
    CatFancier.findById({ _id: id }, (err, data) => {
        if (err) {
            console.log(err);
        } else {
            let name = data.name;
            let age = data.age;
            let id = data._id;
            let fci = data.favoriteCatImg;
            let user_id = data.user_id;
            GamePlayer.findOne({ cat_id : id }, (err, data) => {
                if (err) {
                    console.log(err);
                } else {
                    if (data) {
                        let player_id = data._id;
                        let score = data.score;
                        let times_played = data.times_played;
                        res.render('layouts/site/rest-of-site.ejs', { name, id, age, fci, user_id, player_id,
score, times_played });
                    } else if (!data) {
                        let gamePlayer = new GamePlayer({
                            cat_id : id,
                            user_id : user_id
                        });
                        gamePlayer.save((err, data) => {
```

```
                    if (err) {
                        console.log(err);
                    } else {
                        let player_id = data._id;
                        let score = data.score;
                        let times_played = data.times_played;
                        res.render('layouts/site/rest-of-site.ejs', { name, id, age, fci, user_id,
player_id, score, times_played });
                    }
                });
            }
        }
    });
  }
});


router.post('/game', ensureAuthenticated, (req, res) => {
  let id = req.body.id;
  CatFancier.findById({ _id: id }, (err, data) => {
    if (err) {
      console.log(err);
    } else {
      let name = data.name;
      let age = data.age;
      let id = data._id;
      let fci = data.favoriteCatImg;
      let user_id = data.user_id;
      GamePlayer.findOne({ cat_id : id }, (err, data) => {
        if (err) {
          console.log(err);
        } else {
          if (data) {
            let player_id = data._id;
            let score = data.score;
            let times_played = data.times_played;
            res.render('layouts/site/rest-of-site.ejs', { name, id, age, fci, user_id, player_id,
score, times_played });
          } else if (!data) {
            let gamePlayer = new GamePlayer({
              cat_id : id,
              user_id : user_id
```

```
                    });
                    gamePlayer.save((err, data) => {
                        if (err) {
                            console.log(err);
                        } else {
                            let player_id = data._id;
                            let score = data.score;
                            let times_played = data.times_played;
                            res.render('layouts/site/rest-of-site.ejs', { name, id, age, fci, user_id,
player_id, score, times_played });
                        }
                    });
                }
            }
        });
    }
  });
});

router.get('/howTo/:id', ensureAuthenticated, (req, res) => {
    let id = req.params.id;
    CatFancier.findById({ _id: id }, (err, data) => {
        if (err) {
            console.log(err);
        } else {
            let name = data.name;
            let age = data.age;
            let id = data._id;
            let fci = data.favoriteCatImg;
            let user_id = data.user_id;
            res.render('layouts/site/how-to.ejs', { name, id, age, fci, user_id });
        }
    });
});

router.post('/updateScore/:id', (req, res) => {
    let playerId = req.params.id;
    let newScore = req.body.score;
    GamePlayer.findByIdAndUpdate({ _id : playerId }, { $set:{ score : newScore } }, { new: true },
(err,data) => {
        if (err) {
            console.log(err);
        } else {
```

```
            let cat_id = data.cat_id;
            res.redirect('/restOfSite/game/' + cat_id);
        }
    });
});


module.exports = router;
```

````
* Now your program should keep track of the score. No matter how bad it is, it will never go away! So, let's make a button to reset the score if we want to (we can keep the score if it is good, however!) .
* In the HTML in rest-of-site.ejs shown above the button is already made.
* In app.js make a variable of it, using document.getElementById() and add an addEventListener() to it.
* In the event listener set the score to 0 and update the score.

```
const resetBtn = document.getElementById("resetBtn");

resetBtn.addEventListener('click', () => {
    score = 0;
    updateScore(score);
});
```

````
* The database should be updated, and the score should show as 0.
* app.js should now look like this:

```
const idDiv = document.getElementById("id");
const id = idDiv.getAttribute("data-value");
const playerIdDiv = document.getElementById("player_id");
const playerId = playerIdDiv.getAttribute('data-value');
const scoreDBDiv = document.getElementById("score");
const resetBtn = document.getElementById("resetBtn");
const timesPlayedDiv = document.getElementById("times_played");
const timesPlayed = timesPlayedDiv.getAttribute('data-value');
console.log("timesPlayed: " + timesPlayed);
const ageDiv = document.getElementById("age");
const age = ageDiv.getAttribute('data-value');
const winsTextP = document.getElementById("winsText");
```

```
const lossesTextP = document.getElementById("lossesText");
const target1 = document.getElementById("target1");
const target2 = document.getElementById("target2");
const target3 = document.getElementById("target3");
const playerNumber = document.getElementById("playerNumber");
const targetNumber = document.getElementById("targetNumber");
const randomNum = (max, min) => Math.floor(Math.random() * (max - min + 1)) + min;
const makeRange = (age) => {
    if (age < 10) {
        return randomNum(10, 1);
    } else if (age < 20) {
        return randomNum(20, 1);
    } else {
        return randomNum(100, 1);
    }
};

let score = scoreDBDiv.getAttribute('data-value');
let randomNumber = 0;
let wins = 0;
let losses = 0;
let val1 = 0;
let val2 = 0;
let val3 = 0;

document.addEventListener('DOMContentLoaded', () => {
    setUpGame();
    gameNumbers();
});

resetBtn.addEventListener('click', () => {
    score = 0;
    updateScore(score);
});

function setUpGame() {
    targetNum = 0;
    playerNum = 0;
    randomNumber = randomNum(20, 1);
    targetNum = age * randomNumber;
    val1 = makeRange(age);
    val2 = makeRange(age);
    val3 = makeRange(age);
```

```
    winsTextP.innerText = "Wins: " + wins;
    lossesTextP.innerText = "Losses: " + losses;
    scoreDBDiv.innerText = "Score: " + score;
    playerNumber.innerText = "Player Number: " + playerNum;
    targetNumber.innerText = "Game Number: " + targetNum;
    target1.innerText = val1;
    target2.innerText = val2;
    target3.innerText = val3;
}

function gameNumbers() {
    target1.addEventListener('click', () => {
        play(val1);
    });
    target2.addEventListener('click', () => {
        play(val2);
    });
    target3.addEventListener('click', () => {
        play(val3);
    });
}

function play(val) {
    playerNum = playerNum + val;
    playerNumber.innerText = "Player Number: " + playerNum;
    match(playerNum, targetNum);
}

function match(playerNum, targetNum) {
    if (playerNum >= targetNum + 1) {
        lose();
    }
    if (playerNum === targetNum) {
        win();
    }
}

function lose() {
    losses++;
    lossesTextP.innerText = "Losses: " + losses;
    matchScores();
    setUpGame();
}
```

```
function win() {
    wins++;
    winsTextP.innerText = "Wins: " + wins;
    matchScores();
    setUpGame();
}

function matchScores() {
    if (losses === 3) {
        score--;
        updateScore(score);
        wins = 0;
        losses = 0;
        winsTextP.innerText = "Wins: " + wins;
        lossesTextP.innerText = "Losses: " + losses;
    } else if (wins === 3) {
        score++;
        updateScore(score);
        wins = 0;
        losses = 0;
        winsTextP.innerText = "Wins: " + wins;
        lossesTextP.innerText = "Losses: " + losses;
    }
}

function updateScore(score) {
    let newScore = score;
    return axios.post('/restOfSite/game/game/updateScore/' + playerId, {
        score: newScore
    })
    .then((data) => {
        if (data.status === 200){
            return scoreDBDiv.innerText = "Score: " + score;
        } else {
            console.log("there has been a problem");
        }
    })
    .catch((err) => console.log(err));
}

````

What next?

* This app is almost complete. What is missing is a button that tells the user how many times he/she has played the game.