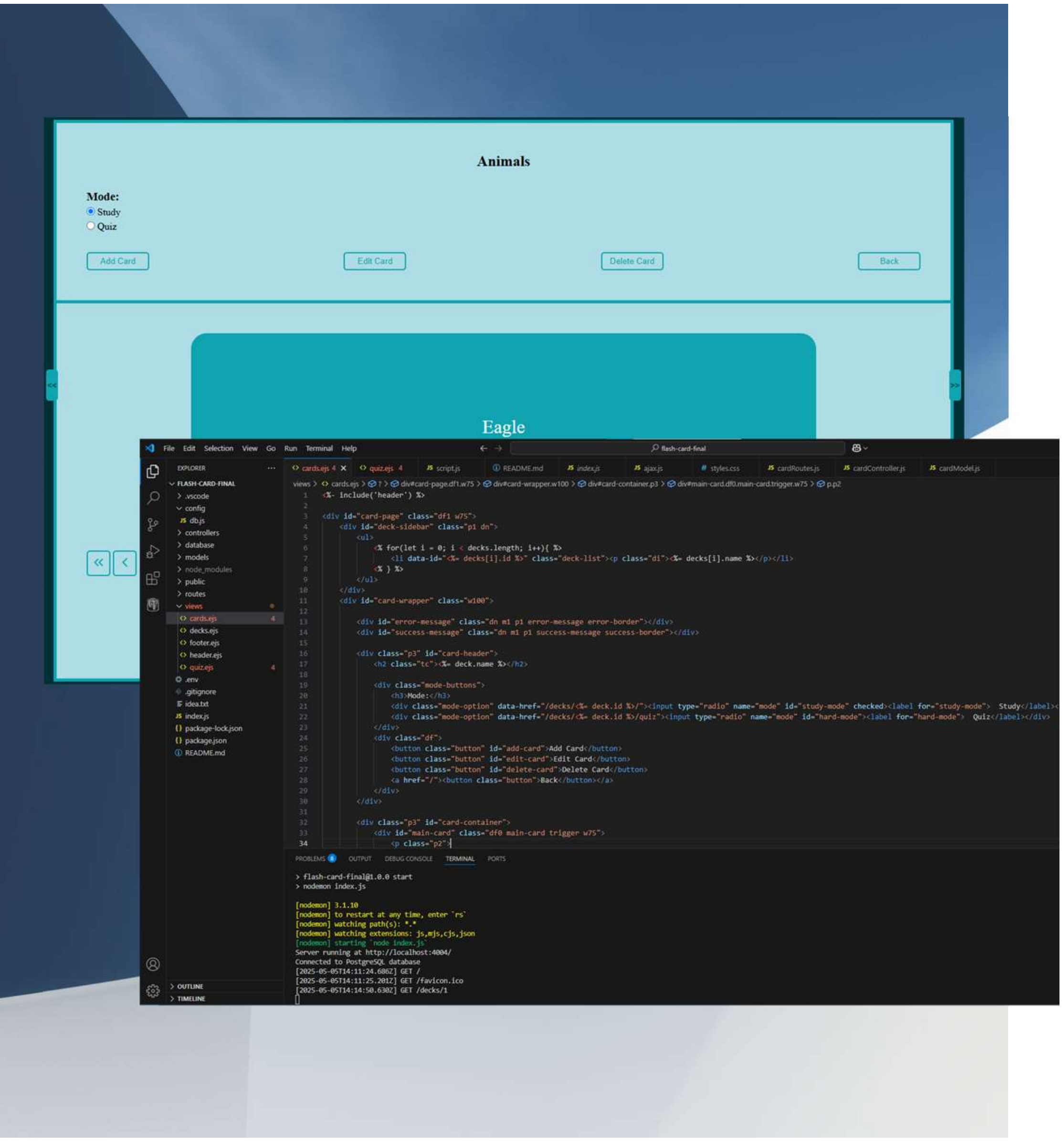


```
<%- include('header') %>

<div id="card-page" class="df1 w75">
    <div id="deck-sidebar" class="p1 dn">
        <ul>
            <% for(let i = 0; i < decks.length; i++){ %>
                <li data-id="<%= decks[i].id %>" class="deck-list"><p class="di"><%= decks[i].name %></p></li>
            <% } %>
        </ul>
    </div>
    <div id="card-wrapper" class="w100">

        <div id="error-message" class="dn m1 p1 error-message error-border"></div>
        <div id="success-message" class="dn m1 p1 success-message success-border"></div>

        <div class="p3" id="card-header">
            <h2 class="tc"><%= deck.name %></h2>

            <div class="mode-buttons">
                <h3>Mode:</h3>
                <div class="mode-option" data-href="/decks/<%= deck.id %>/"><input type="radio" name="mode" id="study-mode" checked><label for="study-mode">  Study</label><
                <div class="mode-option" data-href="/decks/<%= deck.id %>/quiz"><input type="radio" name="mode" id="hard-mode"><label for="hard-mode">  Quiz</label></div>
            </div>
            <div class="df">
                <button class="button" id="add-card">Add Card</button>
                <button class="button" id="edit-card">Edit Card</button>
                <button class="button" id="delete-card">Delete Card</button>
                <a href="/"><button class="button">Back</button></a>
            </div>
        </div>

        <div class="p3" id="card-container">
            <div id="main-card" class="df0 main-card trigger w75">
                <p class="p2">
```

```
> flash-card-final@1.0.0 start
> nodemon index.js

[nodemon] 3.1.10
[nodemon] to restart at any time, enter `rs`
[nodemon] watching path(s): *.*
[nodemon] watching extensions: js,mjs,cjs,json
[nodemon] starting `node index.js`
Server running at http://localhost:4004/
Connected to PostgreSQL database
[2025-05-05T14:11:24.686Z] GET /
[2025-05-05T14:11:25.201Z] GET /favicon.ico
[2025-05-05T14:14:50.630Z] GET /decks/1
```

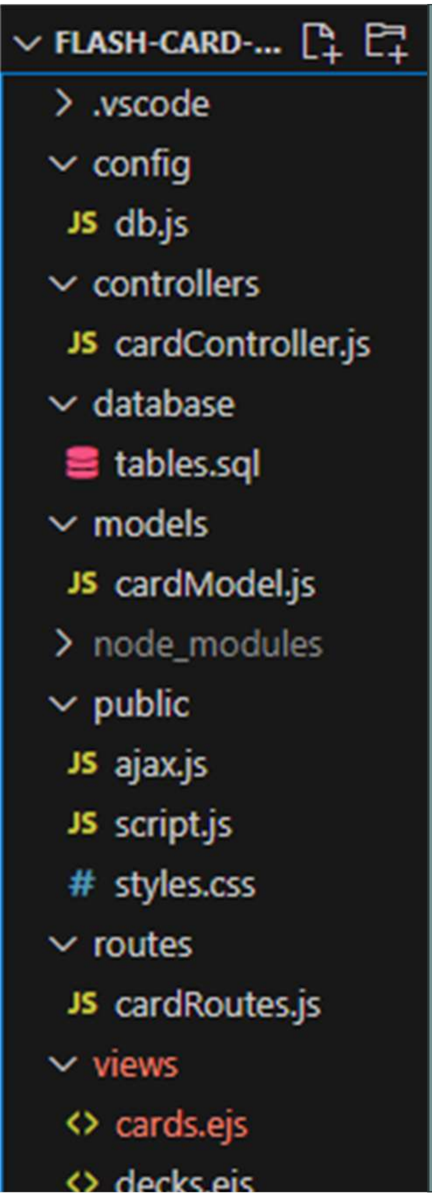
*Presentation for*

# FLASHCARD APP

*By: Joseph Koop*

*For: GUI Programming*

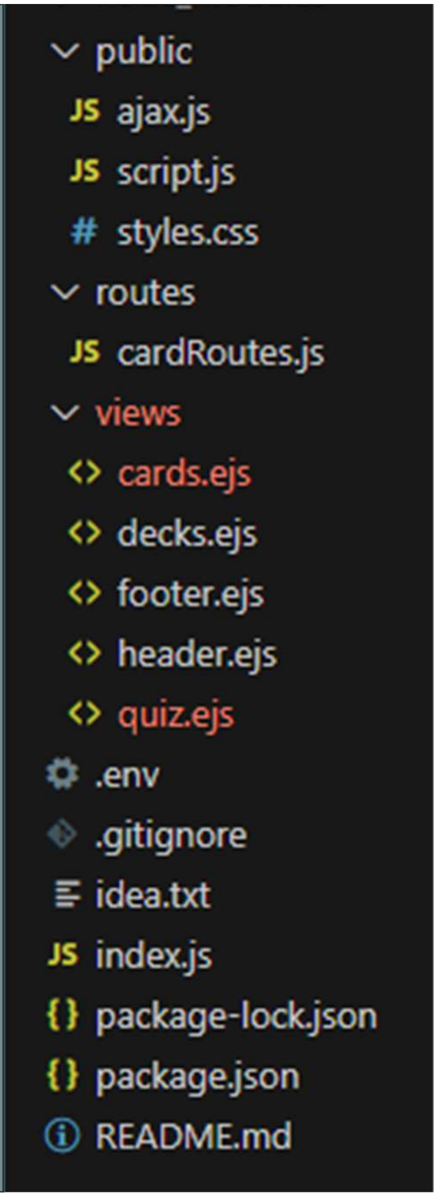
## PART 1: THE STACK



```
FLASH-CARD-...
  > .vscode
  v config
      db.js
  v controllers
      cardController.js
  v database
      tables.sql
  v models
      cardModel.js
  > node_modules
  v public
      ajax.js
      script.js
      styles.css
  v routes
      cardRoutes.js
  v views
      cards.ejs
      decks.ejs
```

## NODE JS

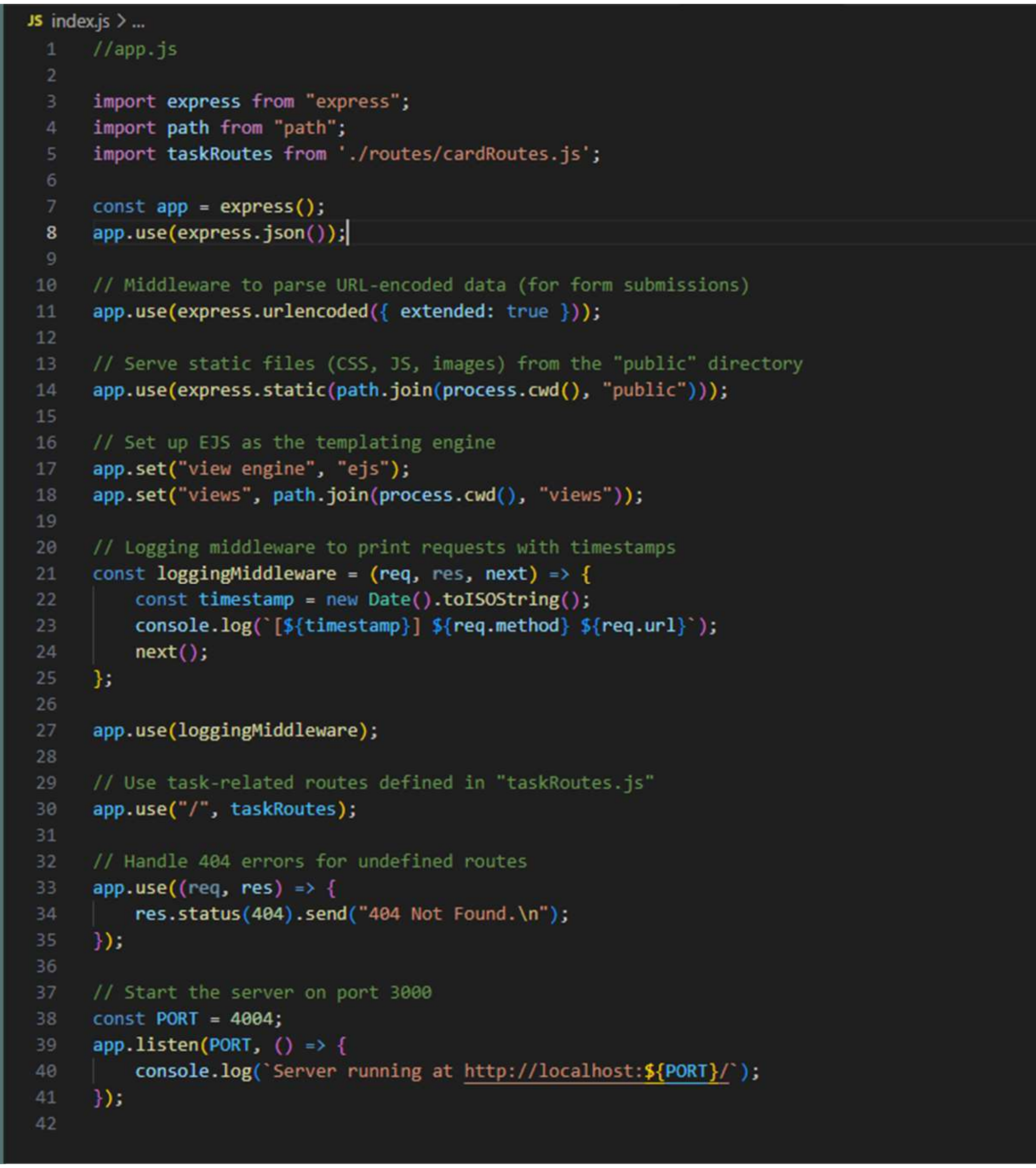
- Created a new Node folder with npm init
    - Installed necessary packages
- Setup file structure to follow MVC model



```
  v public
      ajax.js
      script.js
      styles.css
  v routes
      cardRoutes.js
  v views
      cards.ejs
      decks.ejs
      footer.ejs
      header.ejs
      quiz.ejs
    .env
    .gitignore
    idea.txt
    index.js
    package-lock.json
    package.json
    README.md
```

## EXPRESS

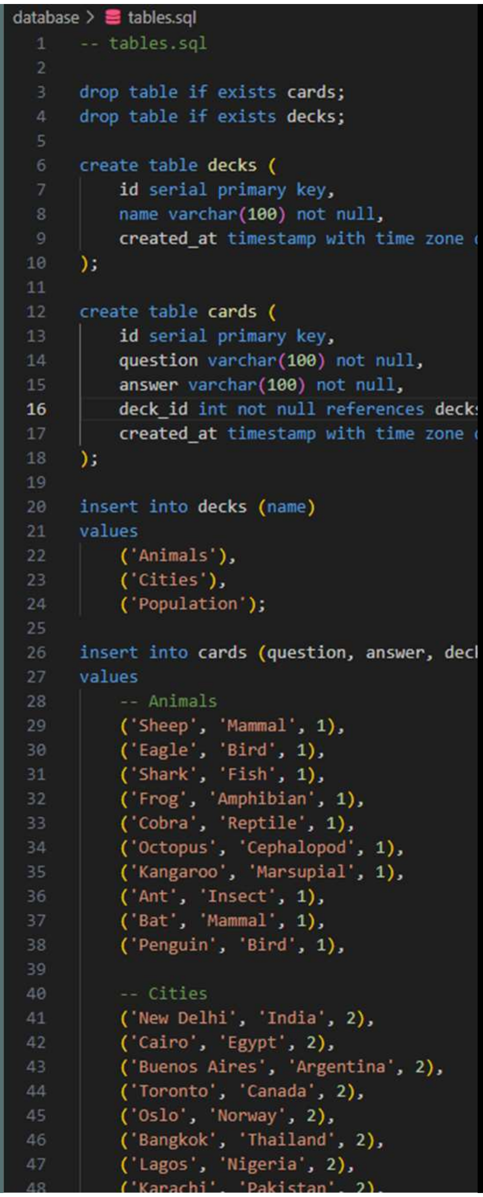
- Created a new express server
- Served static files from public
- Set the view engine to EJS Templating
- Linked my routes file
- Logged requests and handled 404's


JS index.js > ...

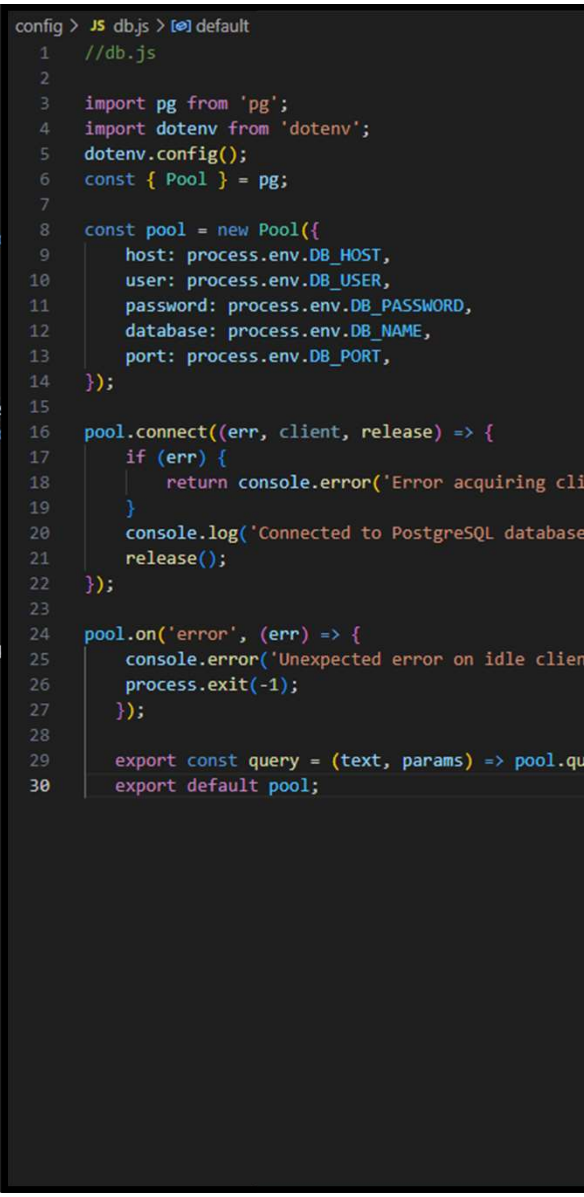
```javascript
//app.js

import express from "express";
import path from "path";
import taskRoutes from './routes/cardRoutes.js';

const app = express();
app.use(express.json());

// Middleware to parse URL-encoded data (for form submissions)
app.use(express.urlencoded({ extended: true }));

// Serve static files (CSS, JS, images) from the "public" directory
app.use(express.static(path.join(process.cwd(), "public")));

// Set up EJS as the templating engine
app.set("view engine", "ejs");
app.set("views", path.join(process.cwd(), "views"));

// Logging middleware to print requests with timestamps
const loggingMiddleware = (req, res, next) => {
    const timestamp = new Date().toISOString();
    console.log(`[${timestamp}] ${req.method} ${req.url}`);
    next();
};

app.use(loggingMiddleware);

// Use task-related routes defined in "taskRoutes.js"
app.use("/", taskRoutes);

// Handle 404 errors for undefined routes
app.use((req, res) => {
    res.status(404).send("404 Not Found.\n");
});

// Start the server on port 3000
const PORT = 4004;
app.listen(PORT, () => {
    console.log(`Server running at http://localhost:${PORT}/`);
});
```

# POSTGRESQL

- Created a Postgres database and user from the terminal
- Ran an SQL file to create and seed the tables
- Stored database info in .env file
- Setup the connection pool in db.js file


database > tables.sql

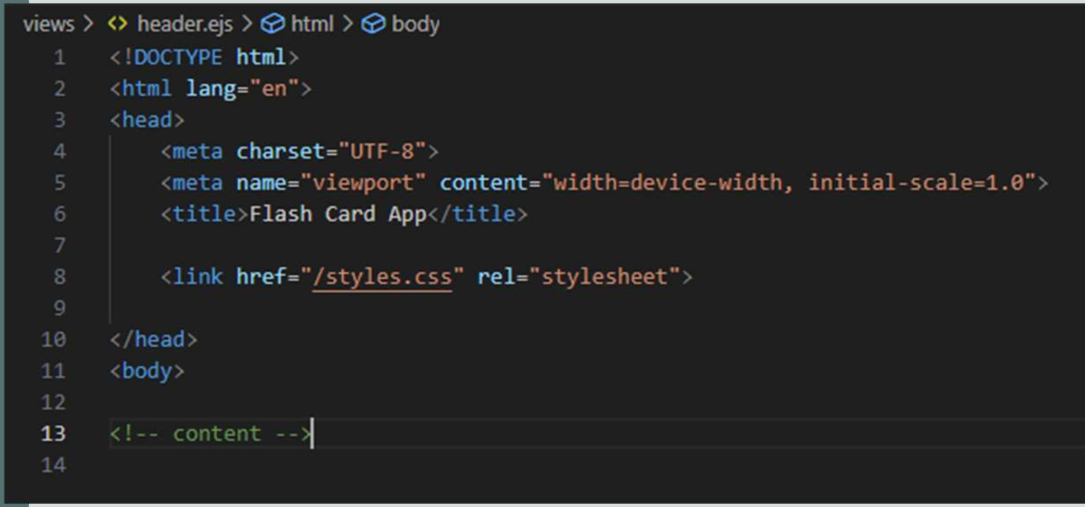
```sql
-- tables.sql

drop table if exists cards;
drop table if exists decks;

create table decks (
    id serial primary key,
    name varchar(100) not null,
    created_at timestamp with time zone
);

create table cards (
    id serial primary key,
    question varchar(100) not null,
    answer varchar(100) not null,
    deck_id int not null references decks
    created_at timestamp with time zone
);

insert into decks (name)
values
    ('Animals'),
    ('Cities'),
    ('Population');

insert into cards (question, answer, decl
values
    -- Animals
    ('Sheep', 'Mammal', 1),
    ('Eagle', 'Bird', 1),
    ('Shark', 'Fish', 1),
    ('Frog', 'Amphibian', 1),
    ('Cobra', 'Reptile', 1),
    ('Octopus', 'Cephalopod', 1),
    ('Kangaroo', 'Marsupial', 1),
    ('Ant', 'Insect', 1),
    ('Bat', 'Mammal', 1),
    ('Penguin', 'Bird', 1),

    -- Cities
    ('New Delhi', 'India', 2),
    ('Cairo', 'Egypt', 2),
    ('Buenos Aires', 'Argentina', 2),
    ('Toronto', 'Canada', 2),
    ('Oslo', 'Norway', 2),
    ('Bangkok', 'Thailand', 2),
    ('Lagos', 'Nigeria', 2),
    ('Karachi', 'Pakistan', 2),
```


config > JS db.js > default

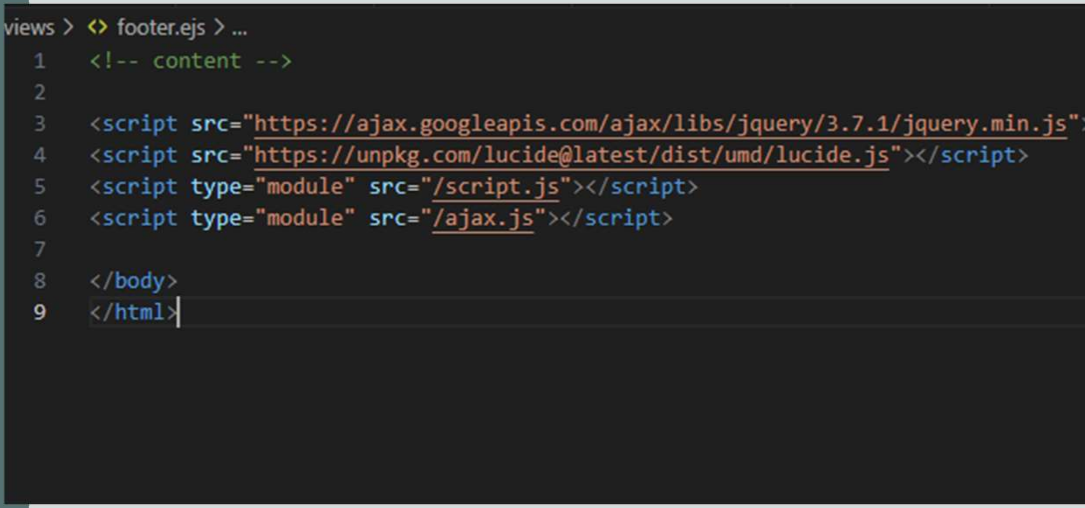
```js
//db.js

import pg from 'pg';
import dotenv from 'dotenv';
dotenv.config();
const { Pool } = pg;

const pool = new Pool({
    host: process.env.DB_HOST,
    user: process.env.DB_USER,
    password: process.env.DB_PASSWORD,
    database: process.env.DB_NAME,
    port: process.env.DB_PORT,
});

pool.connect((err, client, release) => {
    if (err) {
        return console.error('Error acquiring cli
    }
    console.log('Connected to PostgreSQL database
    release();
});

pool.on('error', (err) => {
    console.error('Unexpected error on idle clie
    process.exit(-1);
  });

  export const query = (text, params) => pool.qu
  export default pool;
```

## EJS TEMPLATING

- Used header and footer files that stored the CSS and JS links
- Served the views from the corresponding controller function
- Passed in the related data as props

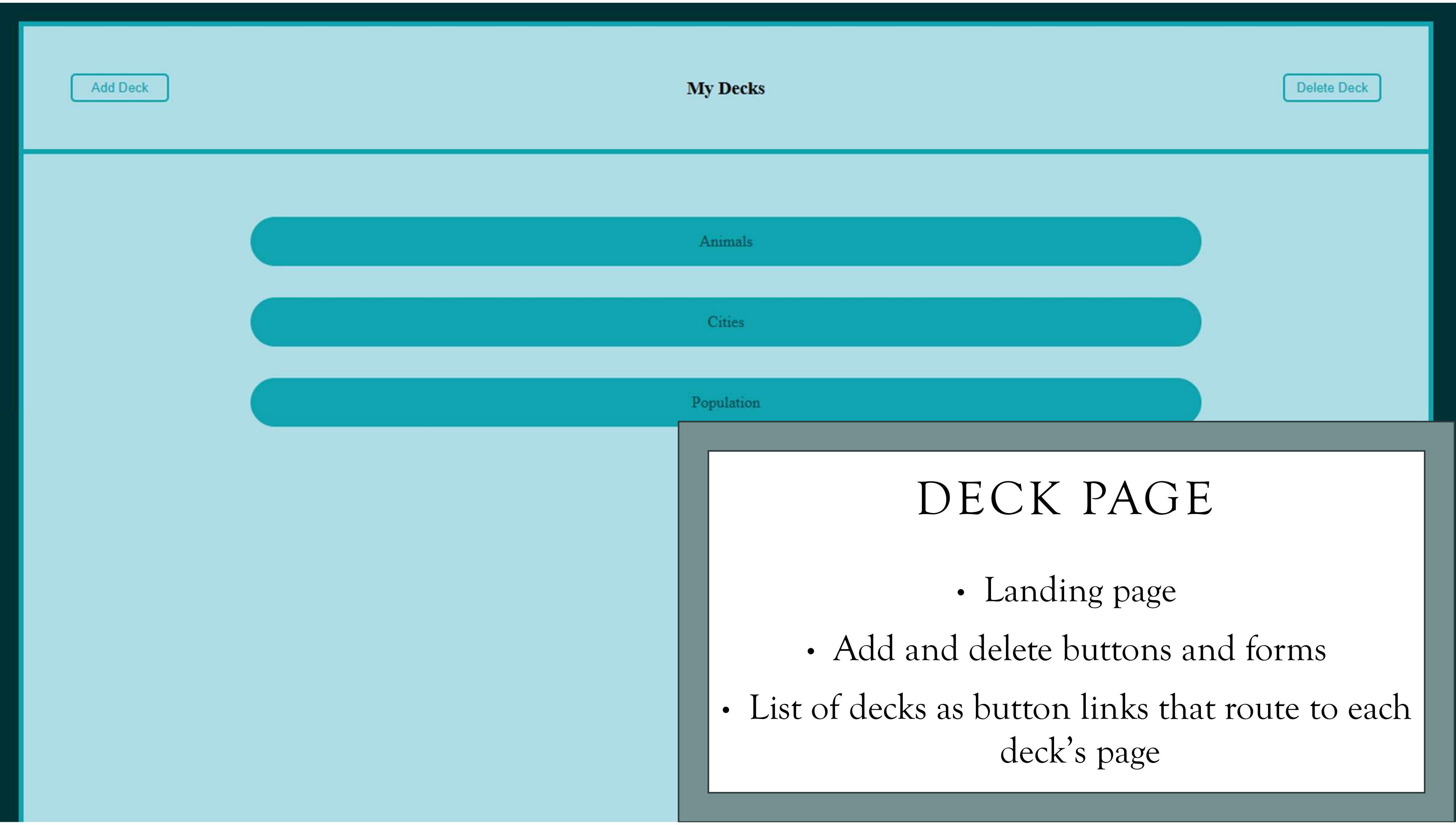

views > header.ejs > html > body

```html
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Flash Card App</title>

    <link href="/styles.css" rel="stylesheet">

</head>
<body>

<!-- content -->

```


views > footer.ejs > ...

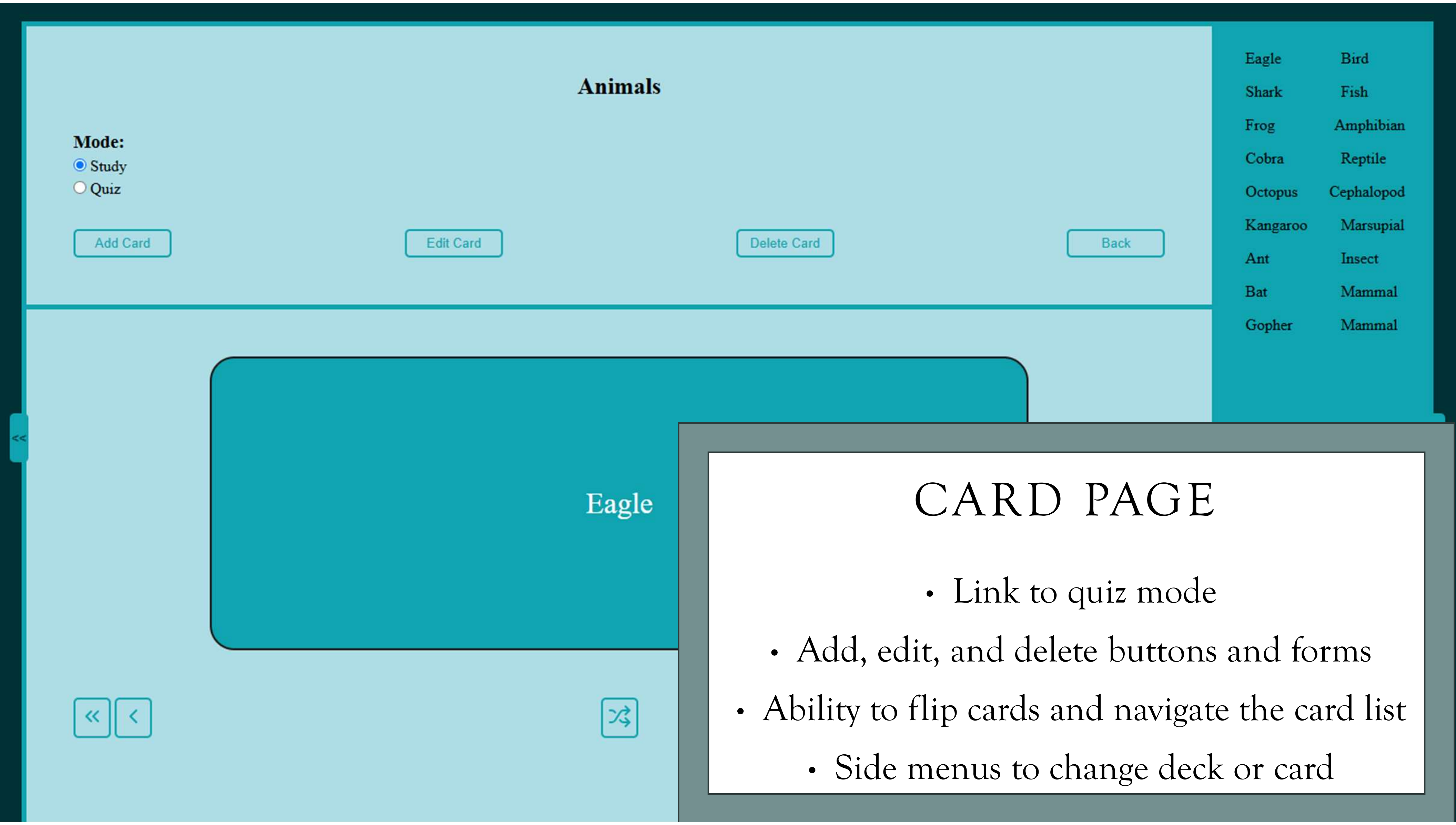
```html
<!-- content -->

<script src="https://ajax.googleapis.com/ajax/libs/jquery/3.7.1/jquery.min.js"
<script src="https://unpkg.com/lucide@latest/dist/umd/lucide.js"></script>
<script type="module" src="/script.js"></script>
<script type="module" src="/ajax.js"></script>

</body>
</html>
```

## PART 2: THE FRONT END



## DECK PAGE

- Landing page
- Add and delete buttons and forms
- List of decks as button links that route to each deck's page



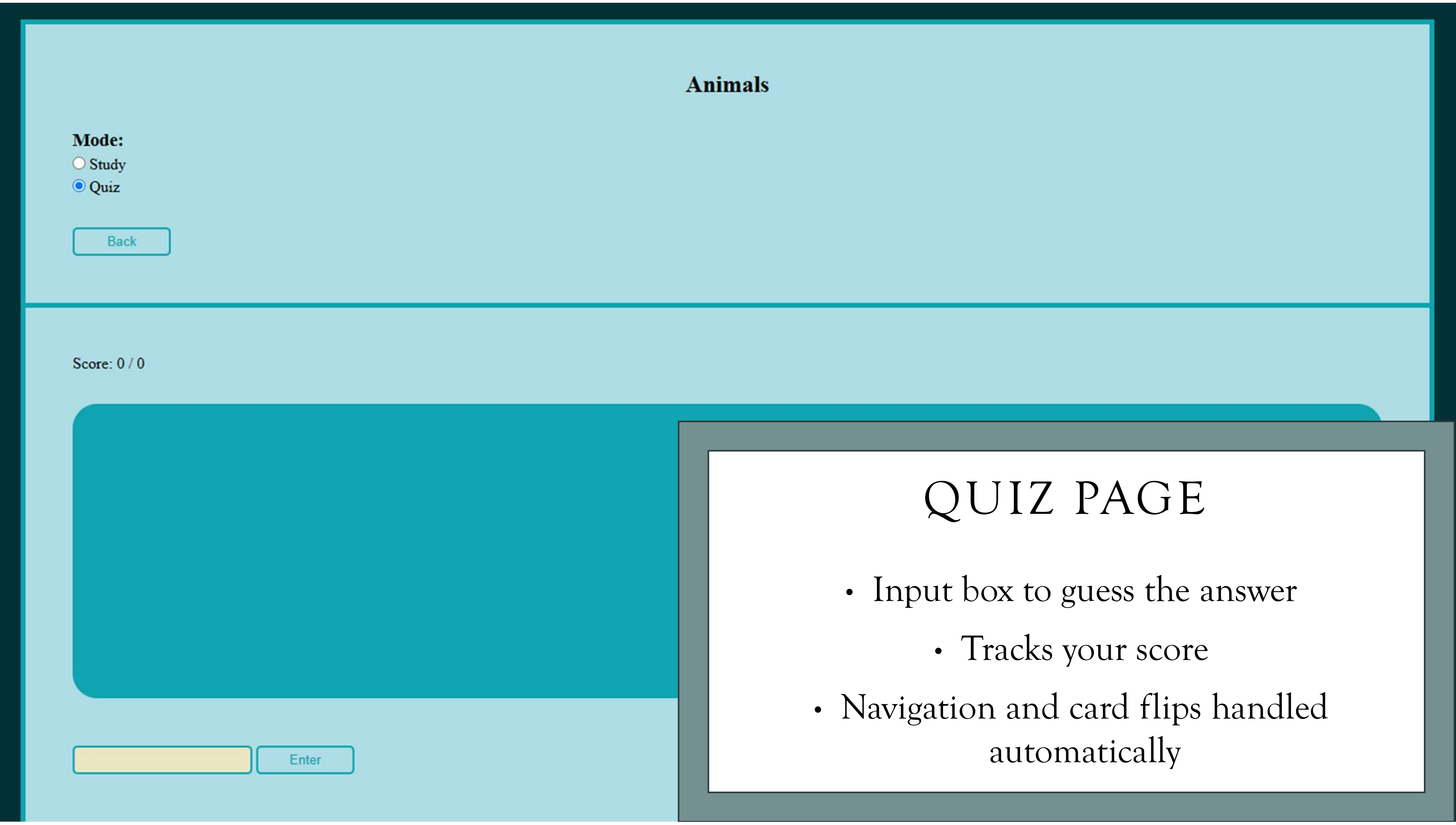
**Animals**

**Mode:**
[✓ Study]
Quiz

| | |
|---|---|
| Eagle | Bird |
| Shark | Fish |
| Frog | Amphibian |
| Cobra | Reptile |
| Octopus | Cephalopod |
| Kangaroo | Marsupial |
| Ant | Insect |
| Bat | Mammal |
| Gopher | Mammal |

## CARD PAGE

- Link to quiz mode
- Add, edit, and delete buttons and forms
- Ability to flip cards and navigate the card list
- Side menus to change deck or card



Animals

Mode:
Study
[✓ Quiz]

Score: 0 / 0

# QUIZ PAGE

- Input box to guess the answer
- Tracks your score
- Navigation and card flips handled automatically

## CSS + JAVASCRIPT

Used Vanilla CSS:

Created classes to style the pages

Found a free icon library to use

Took a color scheme from online

Used Vanilla JavaScript:

Created the hide/show functionality of forms, menus, and messages.

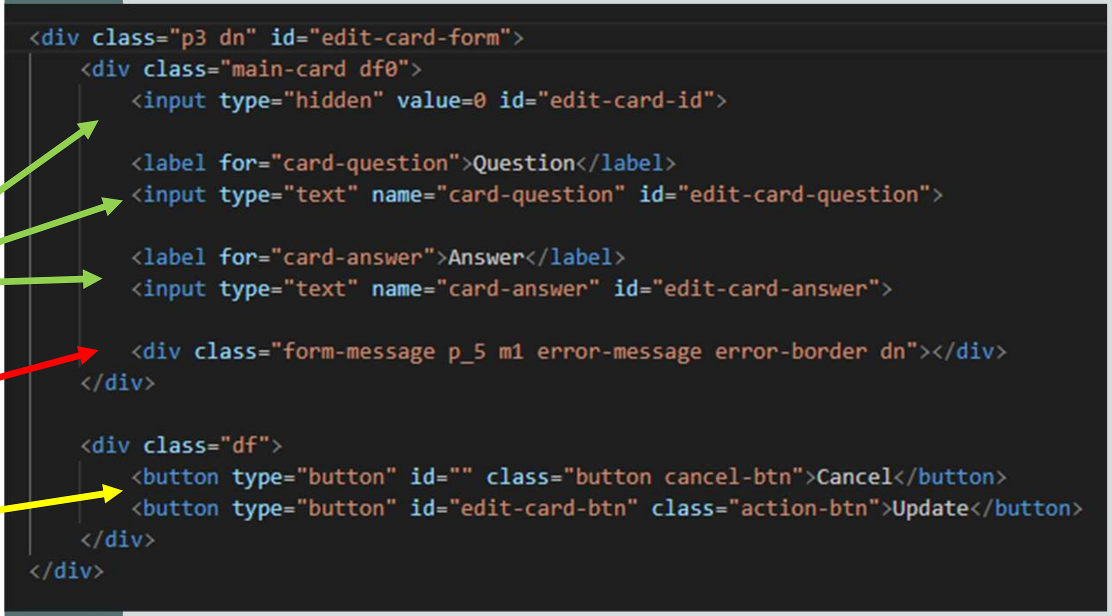
Built the navigation system

Built the quiz score-tracking system

## PART 3: THE BACK END

## FORMS

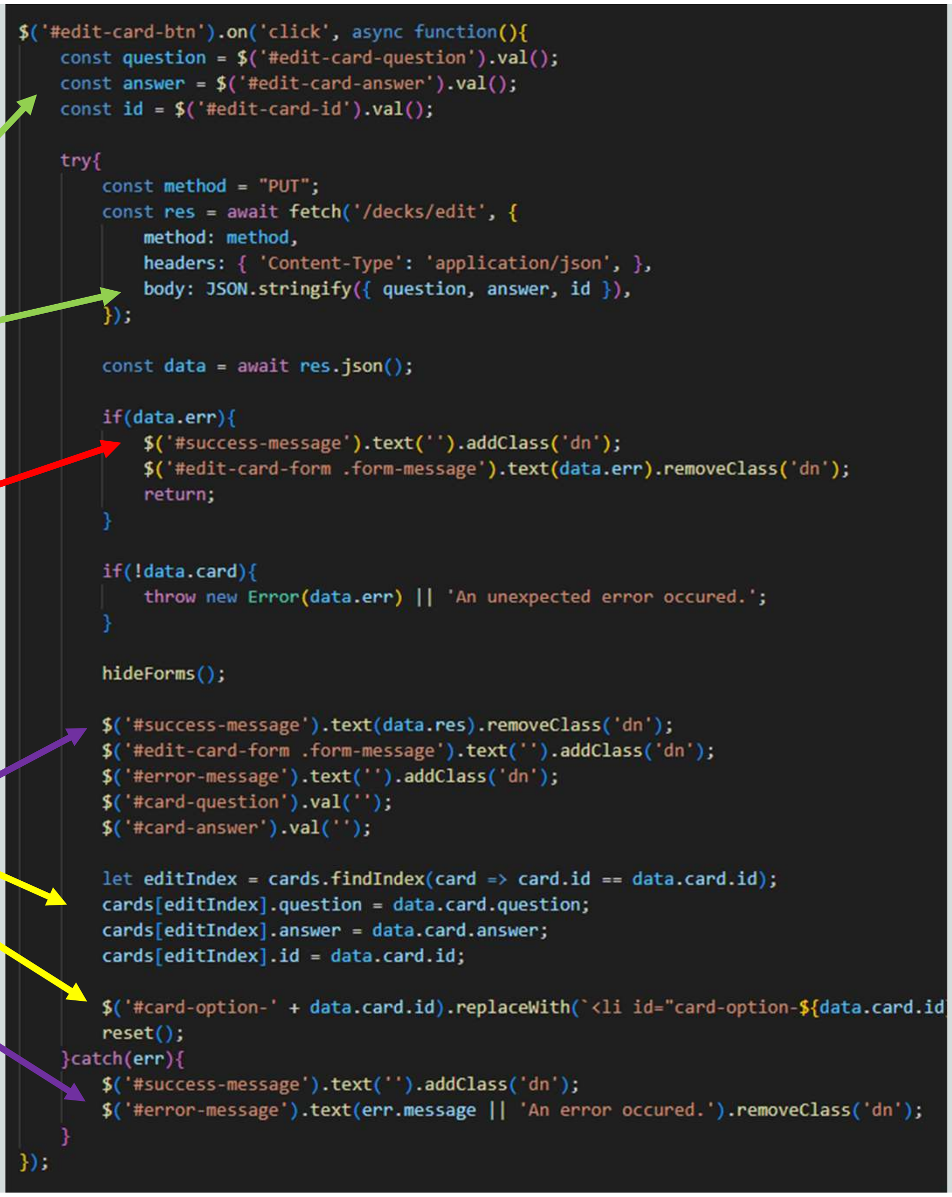
- Started backend logic with forms
- Included necessary fields
- Included section for error messages
- Included handle buttons



```
<div class="p3 dn" id="edit-card-form">
    <div class="main-card df0">
        <input type="hidden" value=0 id="edit-card-id">

        <label for="card-question">Question</label>
        <input type="text" name="card-question" id="edit-card-question">

        <label for="card-answer">Answer</label>
        <input type="text" name="card-answer" id="edit-card-answer">

        <div class="form-message p_5 m1 error-message error-border dn"></div>
    </div>

    <div class="df">
        <button type="button" id="" class="button cancel-btn">Cancel</button>
        <button type="button" id="edit-card-btn" class="action-btn">Update</button>
    </div>
</div>
```

# AJAX

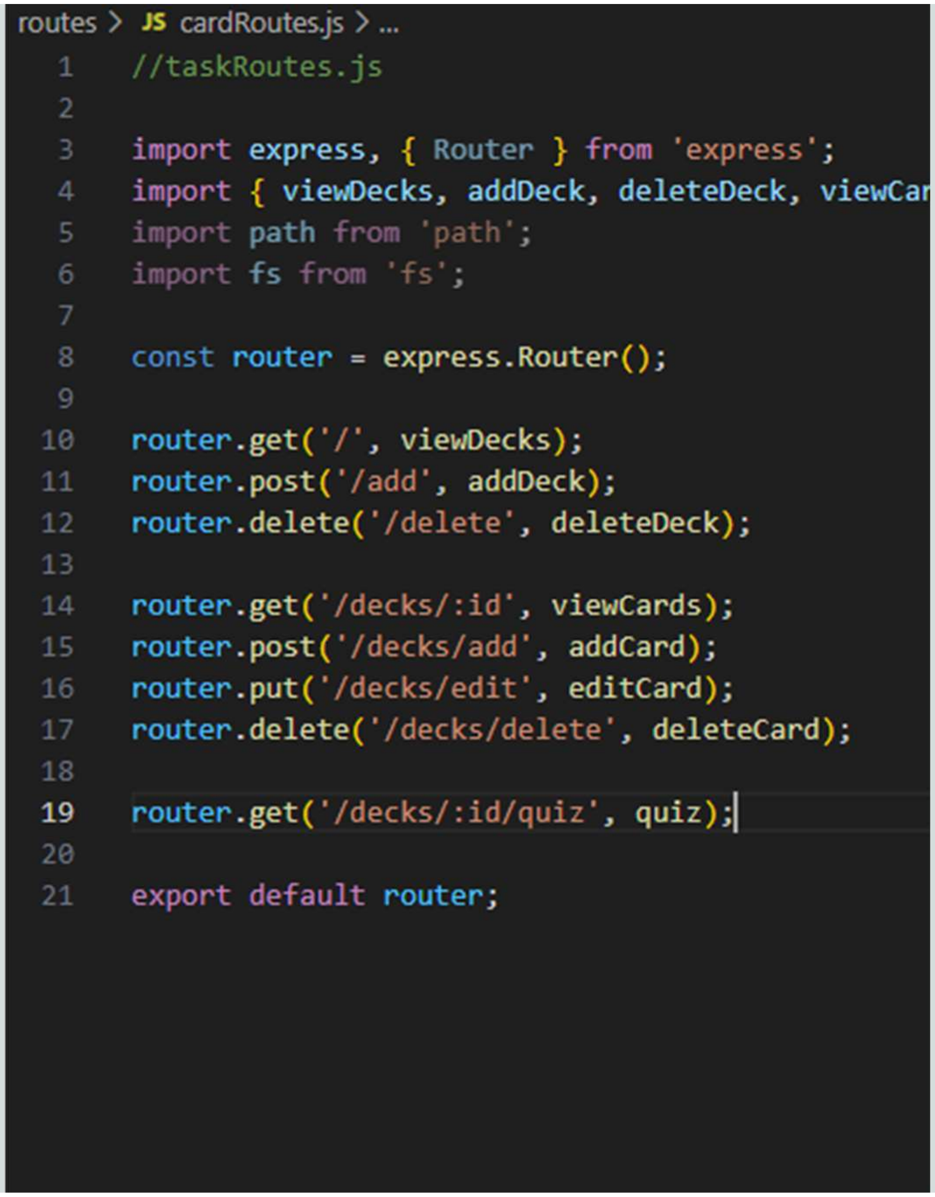
- Got data from forms and passed it to the route
- Handled controlled controller errors by updating the error field in the form
- Updated the DOM if successful
- Displayed uncontrolled errors or success messages at top of page



```
$('#edit-card-btn').on('click', async function(){
    const question = $('#edit-card-question').val();
    const answer = $('#edit-card-answer').val();
    const id = $('#edit-card-id').val();

    try{
        const method = "PUT";
        const res = await fetch('/decks/edit', {
            method: method,
            headers: { 'Content-Type': 'application/json', },
            body: JSON.stringify({ question, answer, id }),
        });

        const data = await res.json();

        if(data.err){
            $('#success-message').text('').addClass('dn');
            $('#edit-card-form .form-message').text(data.err).removeClass('dn');
            return;
        }

        if(!data.card){
            throw new Error(data.err) || 'An unexpected error occured.';
        }

        hideForms();

        $('#success-message').text(data.res).removeClass('dn');
        $('#edit-card-form .form-message').text('').addClass('dn');
        $('#error-message').text('').addClass('dn');
        $('#card-question').val('');
        $('#card-answer').val('');

        let editIndex = cards.findIndex(card => card.id == data.card.id);
        cards[editIndex].question = data.card.question;
        cards[editIndex].answer = data.card.answer;
        cards[editIndex].id = data.card.id;

        $('#card-option-' + data.card.id).replaceWith(`<li id="card-option-${data.card.id}
        reset();
    }catch(err){
        $('#success-message').text('').addClass('dn');
        $('#error-message').text(err.message || 'An error occured.').removeClass('dn');
    }
});
```

## ROUTES

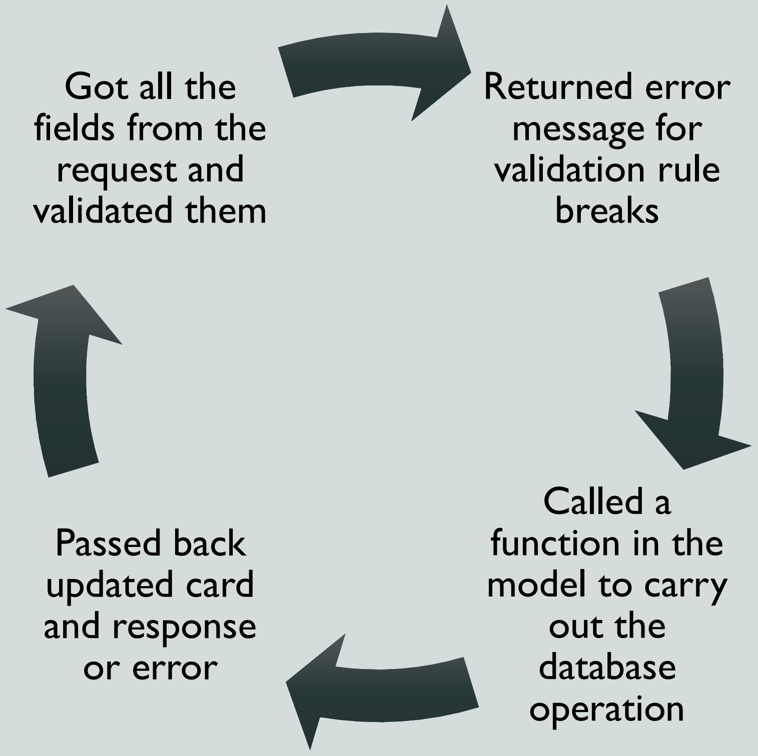
- Created routes for all views
- Created routes for all add/edit/delete functions
- Called function in controller that stores the logic

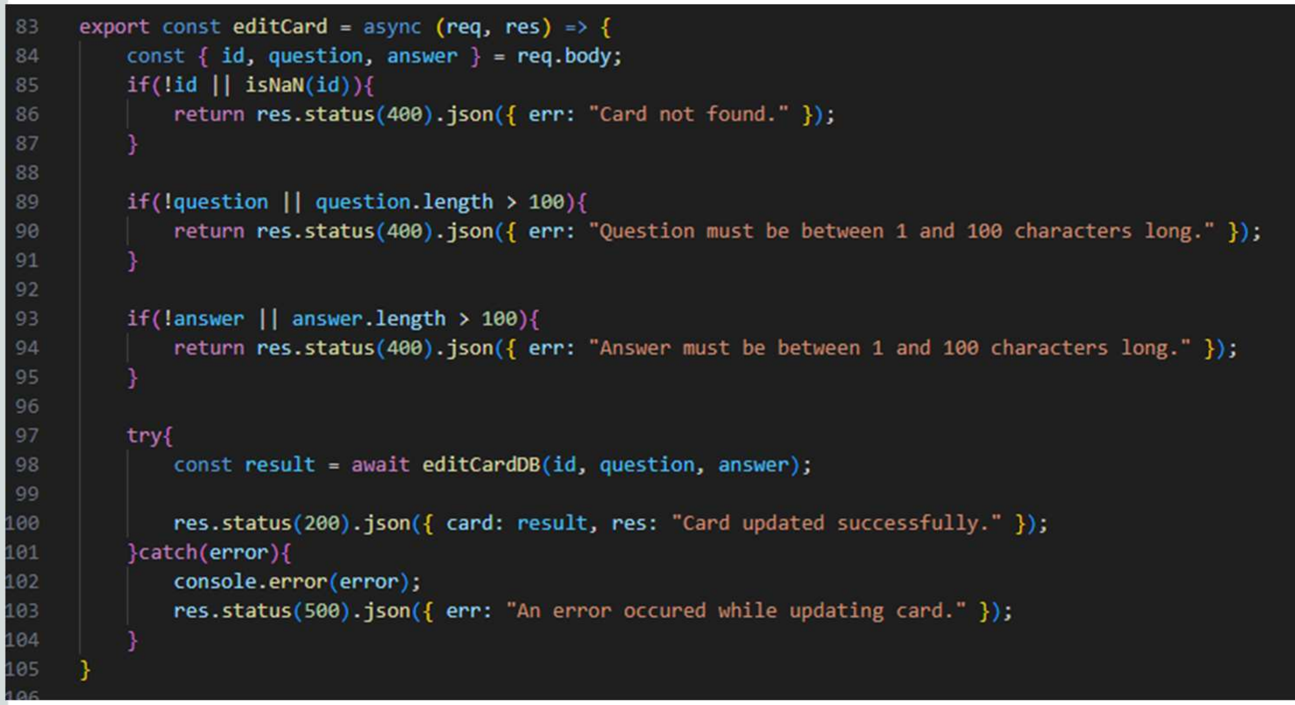


routes > JS cardRoutes.js > ...

```javascript
//taskRoutes.js

import express, { Router } from 'express';
import { viewDecks, addDeck, deleteDeck, viewCar
import path from 'path';
import fs from 'fs';

const router = express.Router();

router.get('/', viewDecks);
router.post('/add', addDeck);
router.delete('/delete', deleteDeck);

router.get('/decks/:id', viewCards);
router.post('/decks/add', addCard);
router.put('/decks/edit', editCard);
router.delete('/decks/delete', deleteCard);

router.get('/decks/:id/quiz', quiz);

export default router;
```

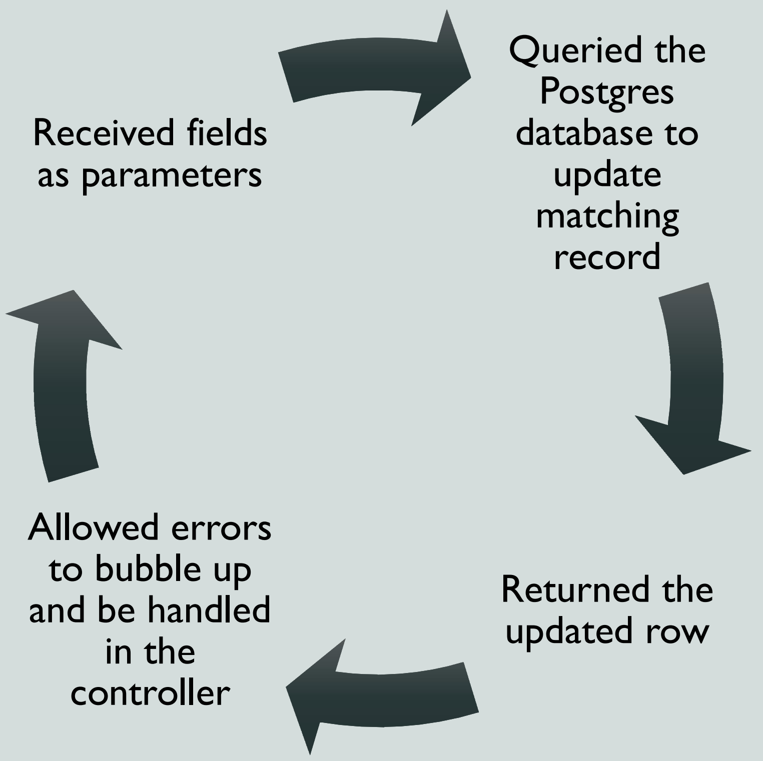
# CONTROLLER

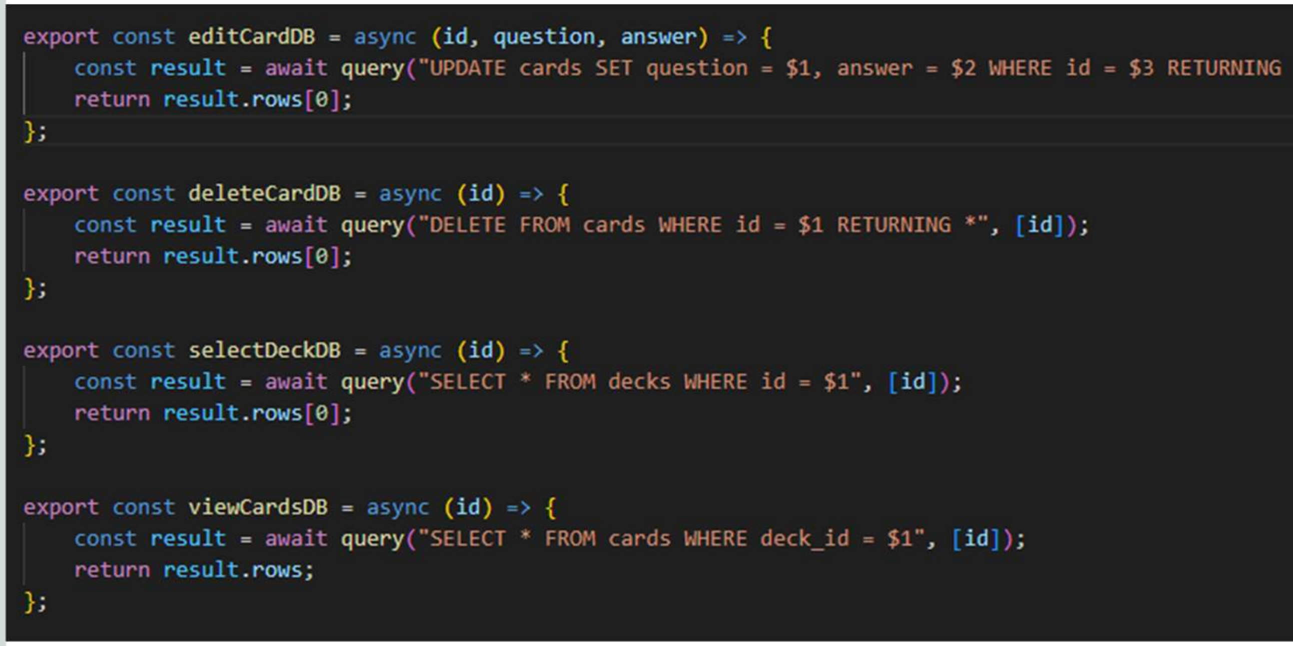

Got all the fields from the request and validated them
Returned error message for validation rule breaks
Called a function in the model to carry out the database operation
Passed back updated card and response or error



```javascript
export const editCard = async (req, res) => {
    const { id, question, answer } = req.body;
    if(!id || isNaN(id)){
        return res.status(400).json({ err: "Card not found." });
    }

    if(!question || question.length > 100){
        return res.status(400).json({ err: "Question must be between 1 and 100 characters long." });
    }

    if(!answer || answer.length > 100){
        return res.status(400).json({ err: "Answer must be between 1 and 100 characters long." });
    }

    try{
        const result = await editCardDB(id, question, answer);

        res.status(200).json({ card: result, res: "Card updated successfully." });
    }catch(error){
        console.error(error);
        res.status(500).json({ err: "An error occured while updating card." });
    }
}
```

# MODEL


Received fields as parameters
Queried the Postgres database to update matching record
Returned the updated row
Allowed errors to bubble up and be handled in the controller



```
export const editCardDB = async (id, question, answer) => {
    const result = await query("UPDATE cards SET question = $1, answer = $2 WHERE id = $3 RETURNING
    return result.rows[0];
};

export const deleteCardDB = async (id) => {
    const result = await query("DELETE FROM cards WHERE id = $1 RETURNING *", [id]);
    return result.rows[0];
};

export const selectDeckDB = async (id) => {
    const result = await query("SELECT * FROM decks WHERE id = $1", [id]);
    return result.rows[0];
};

export const viewCardsDB = async (id) => {
    const result = await query("SELECT * FROM cards WHERE deck_id = $1", [id]);
    return result.rows;
};
```

## PART 4: TESTING

## TESTING + DEBUGGING

Fixed 2 bugs that cost me a log of time:

Array looping issue where quiz mode either didn’t display last card or would go past last card

Class animation issue where moving between cards would disable text flip

Tested every view and every function

Tested edge cases like adding to an empty deck or deleting last card of deck

Tested responsiveness of the design layout

THE END