Lab 5: Camini Server

This is the last lab of the quarter. It's a bit longer than the others but it has a lot of repetition in it.

Deliverable

1. 1 zip file containing:
    a. index.js
    b. package.json
    c. routes/public.js
    d. routes/events.js
    e. admin.js
    f. utils.js
    g. public (you can include the rest, but these are the ones that we're going to work on)
        i. calendar.html
        ii. calendar.css
        iii. calendar.js

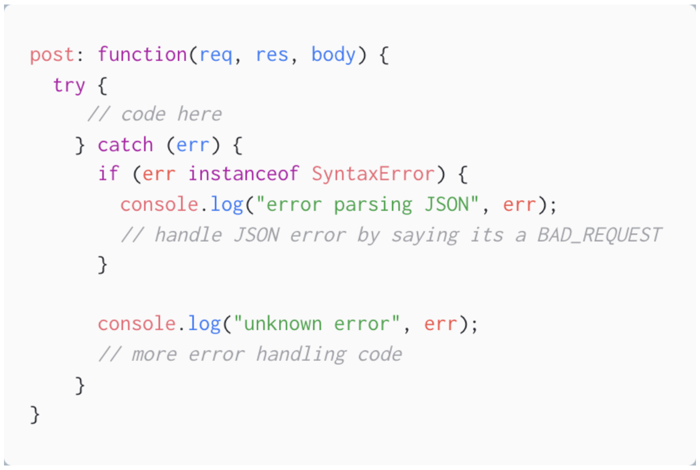
Events Endpoint

1. Create a new file called routes/events.js
2. Create an event object with one property for each month

```
const events = {
  september: {},
  october: {},
  november: {},
  december: {},
};
```

3. Export a single object with 2 functions: get and post (similar to routes/public.js)
    a. Post should take 3 parameters, request, response, and body
    b. When someone makes a get request
        i. use the split function to extract the month request (the URL should look like /api/events/<month>)
        ii. If no month is provided, then send back the JSON string of all the events
        iii. If a month is provided but doesn't exist, make sure to send back an error

        iv. If the month is provided and does exist, send back the JSON string of that month's events
        v. Make sure to use the correct Content-Type, application/json
    c. When someone makes a post request
        i. You'll need to have a try/catch statement (as shown below)
        ii. Parse the body as JSON
            1. Validate that the type, date, and month fields exist.
            2. Month should be one of september, october, november, or december
            3. The type should either be lab, office-hours, quiz, or homework
            4. Make sure to lowercase and trim those strings
            5. If there's an error validating, send back an error code describing why the request is invalid
        iii. If all the information necessary to create an event is present, then add that event into the events object under the correct month and send back a CREATED response
        iv. Use the above information to fill in the code blocks shown below



```
post: function(req, res, body) {
  try {
     // code here
    } catch (err) {
      if (err instanceof SyntaxError) {
        console.log("error parsing JSON", err);
        // handle JSON error by saying its a BAD_REQUEST
      }

      console.log("unknown error", err);
      // more error handling code
    }
}
```

Updated Routing

1. Add a new entry to the routing table for /api/events/. It should call the Events code exported from the previous step
2. Use the utils.readBody function (below) to read the body and pass the body into any POST request.

```
function readBody (request) {
    return new Promise(function (resolve) {
      let body = [];
      request
        .on("data", (chunk) => {
          body.push(chunk);
        })
        .on("end", () => {
          resolve(Buffer.concat(body).toString());
        });
    });
  },
```

Events Script

1. Create a file called lab-5/admin.js
2. You should have a package.json given to you. Put it in your lab-5 folder and run npm install
3. Paste the contents below and then write the createEvents and getEvents functions.
    a. createEvents should send a fetch request (using node-fetch) to send a post request that creates a new event
    b. getEvents should send a fetch request (using node-fetch) to send a get request that reads the events for a month if one is provided. If no month is provided, then it should send a get request for all the events in the month
    c. You'll also have to import the correct libraries. The logic should work out of the box.

```
const sanitize = function (str) {
  return str.trim().toLowerCase();
};

const main = function () {
  const actualArguments = process.argv.slice(2);
  const options = {
    operation: "",
    month: "",
    create: {
      date: "",
      type: "",
    },
  };

  // yes, we want to use ++i because the if statements match the flag
  // the value for that flag will be the next item in the list.
  for (let i = 0; i < actualArguments.length; i++) {
    if (actualArguments[i] === "-o" || actualArguments[i] === "--operation") {
      const operation = sanitize(actualArguments[++i]);
      if (operation !== "create" && operation !== "read") {
        console.log(
          chalk.red(
            `Invalid operation ${operation} provided, must be create or read`
          )
        );
        process.exit(1);
      }
      options.operation = operation;
    } else if (
      actualArguments[i] === "-m" ||
      actualArguments[i] === "--month"
    ) {
      const month = sanitize(actualArguments[++i]);
      if (!utils.VALID_MONTHS.includes(month)) {
        console.log(
          chalk.red(
            `Invalid month ${month} provided, must be one of
${utils.VALID_MONTHS.join(
              " "
            )}`
          )
        );
        process.exit(1);
      }
      options.month = month;
    } else if (actualArguments[i] === "-d" || actualArguments[i] === "--date") {
      const date = parseInt(sanitize(actualArguments[++i]));
      if (date < 1 || date > 31) {
        console.log(
          chalk.red(`Invalid date ${date} provided, must be between 1 and 31`)
        );
        process.exit(1);
      }
      options.create.date = date;
    } else if (actualArguments[i] === "-t" || actualArguments[i] === "--type") {
      const type = sanitize(actualArguments[++i]);
      if (!utils.VALID_TYPES.includes(type)) {
        console.log(
          chalk.red(
            `Invalid type ${type} provided, must be one of
${utils.VALID_TYPES.join(
              " "
            )}`
          )
        );
        process.exit(1);
      }
      options.create.type = type;
    }
```

```
  }

  if (
    options.operation === "create" &&
    (!options.month || !options.create.date || !options.create.type)
  ) {
    console.log(
      chalk.red(
        `Must provide --date, --month, and --type options for the create
operation`
      )
    );
    process.exit(1);
  }

  if (options.operation === "create") {
    return createEvent(options.month, options.create.date, options.create.type);
  } else {
    return getEvents(options.month);
  }
};

main();
```

Updating Calendar.js

In your existing Calendar.js, remove the large EVENTS object. You'll now be receiving that object from the server.

1. Wherever you try to attach the events, create a fetch request to your newly created endpoint by using the URL "/api/events/${month}".
    a. Notice that unlike the fetch functions in the Events script, these functions don't need the address of the server to fetch from. When you fetch something from the same address, you can use an absolute path to indicate that the resource is on the same domain.
2. Use the return value of the fetch call to populate the events.

Grading

The following table shows the point breakdown per section. Each day (including weekends) past the due date for the lab will be -5 points. You must submit your lab on Camino as well as upload it to your website.

Criteria	Points
Events Endpoints	50
Update Routing	15
Events Script	20
Update Calendar.js	15