Department of
COMPUTER SCIENCE
HONG KONG BAPTIST UNIVERSITY

COMP7980 – Dynamic Web and Mobile Programming
COMP7270 – Web and Mobile Programming

Chapter 4 CRUD Operations

Course Instructors: Dr. Ma Shichao, Dr. Zhang Ce, and Mr. Jiang Jintian

Announcement

- Please choose Mac OS

- Find your group mates as soon as possible
  - Send emails to seek group mates

Preparations

- Download lab04-materials from Moodle
- Extract it to a folder
- Get into the folder and install the libraries
  - `cd lab04`
  - `npm install`
- Setup the MongoDB database
  - In the file `utils/db.js`, please assign the variable `process.env.MONGODB_URI` as your own connection string

Route

- A **route** refers to a mechanism or a **mapping** within a web application that **associates a specific URL pattern** with a **handler function** or a controller method. It defines how incoming requests are directed and processed within the server-side application code.
- Routes are typically **associated with specific HTTP methods** (also known as HTTP verbs) in web development. The HTTP methods define the type of operation or action that the client wants to perform on a resource identified by the URL.
- The terms "**endpoint**" and "**route**" are **often used interchangeably** in the context of web development.

Route Handler

```
/* Handle the Form */
router.post('/booking', async function (req, res) {
  const db = await connectToDB();
  try {
    req.body.numTickets = parseInt(req.body.numTickets);
    req.body.terms = req.body.terms? true : false;
    req.body.created_at = new Date();
    req.body.modified_at = new Date();

    let result = await db.collection("bookings").insertOne(req.body);
    res.status(201).json({ id: result.insertedId });
  } catch (err) {
    res.status(400).json({ message: err.message });
  } finally {
    await db.client.close();
  }
});
```

connectToDB() is provided by our db.js

Some data conversions before saving the data to dDB

Common practice to return the id of the newly created document

Route Handler

- req: The **request object** represents the **incoming HTTP request** and **contains various properties and methods** to access different parts of the request.
  - For example, req.body.* contains **key-value pairs** of data submitted in the **request body**, such as a **form being submitted via the** post **method**.
- res: The **response object** represents the **server's response** to the client's request. It provides **methods** and properties to send the response back to the client.
- res.json(response): This method is used to **send data back to the client in JSON format**. It takes **an object** or data structure as an argument and **automatically converts it to JSON** before sending it as the response.

JSON

- JSON (**JavaScript Object Notation**) is a lightweight **data-interchange format** commonly used for transmitting data **between a server and a client**, or between different components of an application.
- It is designed to be **human-readable** and easy to parse for both humans and machines.

A JavaScript Object Array

```
var bookings = [
    {
        email: "martin@choy.com",
        numTickets: 4
    },
    {
        email: "kenny@cheng.com",
        numTickets: 3
    }
];
```

Stringify

Parse

```
[
    {
        "email": "martin@choy.com",
        "numTickets": 4
    },
    {
        "email": "kenny@cheng.com",
        "numTickets": 3
    }
];
```

JSON

List

```
router.get('/booking', async function (req, res) {
    const db = await connectToDB();
    try {
        let results = await db.collection("bookings").find().toArray();
        res.render('bookings', { bookings: results });
    } catch (err) {
        res.status(400).json({ message: err.message });
    } finally {
        await db.client.close();
    }
});
```

- This route handler will retrieve all bookings and renders bookings.ejs.
- A **data object** is provided as the **second argument** of the function.
- In bookings.ejs, the view engine can access the results array using the property name bookings.

Render Method

- router.get('/booking'): This method is used to **render a specific page or template** located in the **views** folder. It allows you to **dynamically generate HTML** or other types of content and send it as a response to the client.
- Please note that the array is referred as bookings here.

bookings.ejs

```
<table>
    <% for (var booking of bookings) { %>

        <tr>
          <td><%= booking.email %></td>
          <td><%= booking.numTickets %></td>
        </tr>

    <% } %>
</table>
```

Route Handler and View

```
router.get('/booking', async function (req, res) {
    const db = await connectToDB();
    try {
        let results = await db.collection("bookings").find().toArray();
        res.render('bookings', { bookings: results });
    } catch (err) {
        res.status(400).json({ message: err.message });
    } finally {
        await db.client.close();
    }
});
```

Visit http://localhost:3000/booking

```
<table>
    <% for (var booking of bookings) { %>

        <tr>
          <td><%= booking.email %></td>
          <td><%= booking.numTickets %></td>
        </tr>

    <% } %>
</table>
```

Representing a booking in the array

Async/await

```
/* Display all Bookings */
router.get('/booking', async function (req, res) {

  let results = await db.collection("bookings").find().toArray();

  res.render('bookings', { bookings: results });

});
```

- The await operator is used here to ensure **the methods resolve** before we move on to the next line.
- "await" can only exist **inside** an async function, so it **would not block other functions** (router handlers), from running concurrently.

Other CRUD operations (Create, Read, Update and Delete)

Display one Booking

- Notice the use of findOne() here. This method returns the **first matching document**.

```
{
  _id: ObjectId("6332ee629f7735181a381f2f"),
  email: 'tony@starks.com',
  numTickets: 2,
  team: 'Avengers',
  superhero: 'Ironman',
  payment: 'Paypal'
}
```

```
/* Display a single Booking */
router.get('/booking/read/:id', async function (req, res) {
  const db = await connectToDB();
  try {
    let result = await db.collection("bookings").findOne({ _id: new ObjectId(req.params.id) });
    if (result) {
      res.render('booking', { booking: result });
    } else {
      res.status(404).json({ message: "Booking not found" });
    }
  } catch (err) {
    res.status(400).json({ message: err.message });
  } finally {
    await db.client.close();
  }
});
```

find by primary key

Path Parameters

- We may want to obtain the **path parameters** of our URL.
- This could be done by setting **dynamic parameters** in a **route**.
- In index.js, develop

```
router.get('/booking/read/:id', async function (req, res) {
```

- In the route handler, these parameters are available under req.params.*

req.params.id

```
http://localhost:3000/booking/read/6332ee629f7735181a381f2f
```

Delete

```
// Delete a single Booking
router.post('/booking/delete/:id', async function (req, res) {
  const db = await connectToDB();
  try {
    let result = await db.collection("bookings").deleteOne({ _id: new ObjectId(req.params.id) });
    if (result.deletedCount > 0) {
      res.status(200).json({ message: "Booking deleted" });
    } else {
      res.status(404).json({ message: "Booking not found" });
    }
  } catch (err) {
    res.status(400).json({ message: err.message });
  } finally {
    await db.client.close();
  }
});
```

req.params.id

```
action="http://localhost:3000/booking/delete/6332ee629f7735181a381f2f"
```

- The deleteOne() method removes the specified document.

Update

- get request:
  returns a **html form** with **pre-filled values**

```
// display the update form
router.get('/booking/update/:id', async function (req, res) {
  const db = await connectToDB();
  try {
    let result = await db.collection("bookings").findOne(
                                  { _id: new ObjectId(req.params.id) });
    if (result) {
      res.render('update', { booking: result });
    } else {
      res.status(404).json({ message: "Booking not found" });
    }
  } catch (err) {
    res.status(400).json({ message: err.message });
  } finally {
    await db.client.close();
  }
});
```

Update

```
// Update a single Booking
router.post('/booking/update/:id', async function (req, res) {
  const db = await connectToDB();
  try {
    req.body.numTickets = parseInt(req.body.numTickets);
    req.body.terms = req.body.terms? true : false;
    req.body.superhero = req.body.superhero || "";
    req.body.modified_at = new Date();

    let result = await db.collection("bookings").updateOne(
                          { _id: new ObjectId(req.params.id) }, { $set: req.body });

    if (result.modifiedCount > 0) {
      res.status(200).json({ message: "Booking updated" });
    } else {
      res.status(404).json({ message: "Booking not found" });
    }
  } catch (err) {
    res.status(400).json({ message: err.message });
  } finally {
    await db.client.close();
  }
});
```

Fallback value for superhero not being submitted

- post request: **updates the data** in the database.

http://localhost:3000/booking/search?email=tony&numTickets=2

Search

```
// Search Bookings
router.get('/booking/search', async function (req, res) {
  const db = await connectToDB();
  try {
    let query = {};
    if (req.query.email) {
      query.email = { $regex: req.query.email };
    }
    if (req.query.numTickets) {
      query.numTickets = parseInt(req.query.numTickets);
    }

    let result = await db.collection("bookings").find(query).toArray();
    res.render('bookings', { bookings: result });
  } catch (err) {
    res.status(400).json({ message: err.message });
  } finally {
    await db.client.close();
  }
});
```

We use $regex here to provide **partial matching**.

Both criteria (if provided) have to be satisfied

Pagination

```
// Pagination based on query parameters page and limit, also returns total number of documents
router.get('/booking/paginate', async function (req, res) {
  const db = await connectToDB();
  try {
    let page = parseInt(req.query.page) || 1;
    let perPage = parseInt(req.query.perPage) || 10;
    let skip = (page - 1) * perPage;

    let result = await db.collection("bookings").find().skip(skip).limit(perPage).toArray();
    let total = await db.collection("bookings").countDocuments();

    res.render('bookings', { bookings: result, total: total, page: page, perPage: perPage });
  } catch (err) {
    res.status(400).json({ message: err.message });
  }
  finally {
    await db.client.close();
  }
});
```

Number of items per page

Number of items to be skipped

page and perPage may have been modified in the route handler

http://localhost:3000/booking/paginate?perPage=2&page=2

Pagination Links

- Utilize the **template engine** to calculate **the total number of pages**.

```
<% for (let i = 1; i <= Math.ceil(total / perPage); i++) { %>
  <% if(i === page) { %>
    <span><%= i %></span>
  <% } else { %>
    <a href="/booking/paginate?page=<%= i %>&perPage=<%= perPage %>"><%= i %></a>
  <% } %>
<% } %>
```

Status Code

- A status code, also known as an **HTTP status code** or **response code**, is a three-digit numeric code that is included in the response sent by a server to **indicate the status or outcome of an HTTP request** made by a client.
- Status codes provide information about the **success**, **failure**, or other conditions related to the request-response cycle. They allow the client and server to communicate and **understand the result of the request without relying solely on the response body**.

Status Code	Meaning
200	OK
201	Created
204	No Content
400	Bad Request
401	Unauthorized
403	Forbidden
404	Not Found
405	Method Not Allowed
500	Internal Server Error
503	Service Unavailable