Lab 1 Integrate MongoDB with Node.js

1. Database Models (/models/ablum.js)

Defined a MongoDB schema for an album object.

The MongoDB schema for an album object includes four properties:

- **_id**: A unique identifier for the album, automatically generated by MongoDB. This property is not included in the request body when creating a new album, as it is generated by MongoDB.
- **title**: The title of the album, a string.
- **artist**: The artist of the album, a string.
- **year**: The year the album was released, an integer.

Additionally, the schema includes a pre-save hook that automatically assigns a unique ID to new albums based on the highest existing ID in the database. This ensures that each album has a unique identifier (_id).

Finally, it exports a Mongoose model for the album, which can be used to query and interact with the album collection in the MongoDB database.

2. Backend (index.js)

The backend is built using Node.js and Express. The following routes were created:

- GET /api/albums: Returns a list of all albums.
- POST /api/albums: Creates a new album.

  It first creates a new object with the album's title, artist, and year taken from the request body, then queries the database to find any existing albums with the same title, artist, and year.

  If any are found, it sends a 409 conflict status response to the client.

If no matching albums are found, it creates a new Album object with the provided data, assigns it an ID of 1 (for simplicity), and saves it to the database.

It then sends a 201 created status response to the client with the newly created album object in the response body.

If an error occurs during this process, it sends a 500 internal server error response with the error message in the response body.

- GET /api/albums/:title: Returns a specific album(s) based on its title.

  It takes in the album title as a parameter in the URL and uses it to query the database for any albums with a matching title.

  If no albums with the given title are found, it sends a 404 not found status response to the client.

  If one or more matching albums are found, it sends a JSON response to the client with the album(s) object(s).

- PUT /api/albums/:id: Updates a specific album based on its ID.

  It takes in the ID of the album to be updated as a parameter in the URL.

  The route first searches the database for the album with the given ID.

  If the album is not found, it sends a 404 not found status response to the client.

  If the album is found, it updates the album's title, artist, and year properties with the values from the request body, saves the updated album to the database, and sends a JSON response to the client with the updated album object.

- DELETE /api/albums/:id: Deletes a specific album based on its ID.

  It takes in the ID of the album to be deleted as a parameter in the URL.

  The route first searches the database for the album with the given ID.

  If the album is not found, it sends a 404 not found status response to the client. If the album is found, it deletes the album from the database and sends a JSON response to the client with a success message.

3. Frontend (client.js & index.html)

3.1 index.html

- includes a table for displaying a list of albums and a form for creating a new album.
- includes a Bootstrap stylesheet and JavaScript bundle, which provide styling and interactivity to the webpage.
- The table with headers for "ID", "Title", "Artist", and "Year". The table has an empty tbody element, which will be populated with album data using JavaScript.
- The form is for creating new albums, which includes fields for "Title", "Artist", and "Year", as well as a "Create" button.

3.2 client.js

Uses fetch() to interact with the backend express server.

- The list of albums is displayed in a table on the webpage. Each album is represented as a table row (`<tr>`) with four table cells (`<td>`): one for the album ID, one for the album title, one for the album artist, and one for the album year. Each album has an `Update` button and a `Delete` button associated with it.
- Defines an `async` function `getAlbums()` that retrieves a list of all albums from the database using a `GET` request to the `/api/albums` endpoint of the backend server.
- Attaches an event listener to the `window` object that calls `getAlbums()` when the page is loaded.
- Attaches an event listener to the table's `tbody` element that listens for click events.

  If the click target is an `Update` button, the script replaces the table cell contents with input fields, allowing the user to update the album information. The `Update` button will be replaced with a `Save` button.

  If the click target is a `Save` button, the updated album information is sent to the backend server using a `PUT` request to the `/api/albums/:id` endpoint, and the table cell contents are updated with the new information. The `Save` button is replaced with an `Update` button.
- Attaches an event listener to each `Delete` button that listens for click events. If the user confirms the deletion, the script sends a `DELETE` request to the `/api/albums/:id` endpoint of the backend server to delete the album from the database. The row containing the album information is removed from the table.