

DATA-236 Sec 21 - Distributed Systems for Data Engineering

HOMEWORK 4

Vimalannadhan Sivanandham
017596436

GitHub -

<https://github.com/Vimalanandhan/DATA-236---Distributed-Systems-for-Data-Engineering/tree/main/Assignments/Assignment%204>

Q1. React (10 Points)

Create a Book Management App where users can add, update, and delete books.

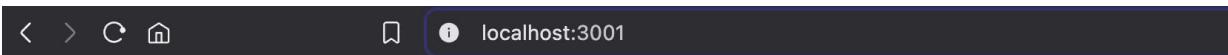
I. Home Page (1 Point)

Write the necessary code to create the ‘Home.jsx’ component which will display all the books. The Home component should be rendered when the user is on the root route ‘/’.



Book Management System

Add New Book



Book Management System

Add New Book

To Kill a Mockingbird

Author: Harper Lee

ISBN: 978-0446310789

Update

Delete

Atomic Habits

Author: James Clear

ISBN: 1234567

Update

Delete


```
# Update.css U # Create.css U JS App.js U X JS index.js U JS app.js M
hw4-template > src > JS App.js > [e] default
  8  const App = () => {
 38    <Router>
 39      <div className="App">
 40        <Routes>
 41          <Route path="/" element={<Home books={books} onError={setError} />} />
 42          <Route path="/create" element={<CreateBook onAddBook={setBooks} />} />
U # Update.css U # Create.css U JS App.js U JS app.js M ☀ Home.jsx U X
hw4-template > src > components > Home > ☀ Home.jsx > [e] Home
  1  import React, { useState, useEffect } from 'react';
  2  import { Link } from 'react-router-dom';
  3  import './Home.css';
  4
  5  const Home = () => {
  6    const [books, setBooks] = useState([]);
  7
  8    useEffect(() => {
  9      // Fetch books from API
 10      fetch('http://localhost:3000/api/books')
 11        .then(response => response.json())
 12        .then(data => setBooks(data))
 13        .catch(error => console.error('Error:', error));
 14    }, []);
 15
 16    return [
 17      <div className="home-container">
 18        <h1>Book Management System</h1>
 19        <Link to="/create" className="add-button">Add New Book</Link>
 20
 21        <div className="books-grid">
 22          {books.map(book => (
 23            <div key={book.id} className="book-card">
 24              <h3>{book.title}</h3>
 25              <p>Author: {book.author}</p>
 26              <p>ISBN: {book.isbn}</p>
 27              <div className="book-actions">
 28                <Link to={`/update/${book.id}`} className="edit-button">Update</Link>
 29                <Link to={`/delete/${book.id}`} className="delete-button">Delete</Link>
 30              </div>
 31            </div>
 32          )})
 33        </div>
 34      </div>
 35    ];
 36  };
 37
 38  export default Home;
```

II. Add a New Book (**2 Points**)

Write the necessary code to create the 'CreateBook.jsx' component. The component should be rendered when the user is on the '/create' route.

The component should accept props (similar to demo) to add the new book and have the following:

- An input field for entering the Book Title.
- An input field for entering the Author Name.
- A submit button labeled "Add Book". When the user clicks the Add book button:
- A new book should be added to the book list with an auto-incremented book ID.
- The user should be redirected to the home page to see the updated book list.

localhost:3001/create

Add New Book

Book Title:

Test Book

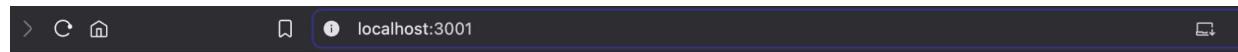
Author Name:

Test Author

ISBN:

987654

Add Book



Book Management System

Add New Book

To Kill a Mockingbird

Author: Harper Lee

ISBN: 978-0446310789

Update

Delete

Atomic Habits

Author: James Clear

ISBN: 1234567

Update

Delete

Test Book

Author: Test Author

ISBN: 987654

Update

Delete

```
>CreateBook.jsx U JS App.js U X
hw4-template > src > JS App.js > ...
8  const App = () => {
33
34    if (loading) return <div>Loading books...</div>;
35    if (error) return <div>Error: {error}</div>;
36
37    return (
38      <Router>
39        <div className="App">
40          <Routes>
41            <Route path="/" element={<Home books={books} onError={setError} />} />
42            <Route path="/create" element={<CreateBook onAddBook={setBooks} />} />
43            <Route
```

⚙️ CreateBook.jsx ✘ × | JS App.js ✘

```
hw4-template > src > components > Create > ⚙️ CreateBook.jsx > ...
```

```
1 import React, { useState } from 'react';
2 import { useNavigate } from 'react-router-dom';
3 import './CreateBook.css';
4
5 const CreateBook = ({ onAddBook }) => {
6   const [title, setTitle] = useState('');
7   const [author, setAuthor] = useState('');
8   const [isbn, setIsbn] = useState('');
9   const [error, setError] = useState(null);
10  const navigate = useNavigate();
11
12  const handleSubmit = async (e) => {
13    e.preventDefault();
14    setError(null);
15
16    try {
17      const response = await fetch('http://localhost:3000/api/books', {
18        method: 'POST',
19        headers: {
20          'Content-Type': 'application/json',
21        },
22        body: JSON.stringify({
23          title,
24          author,
25          isbn
26        }),
27      });
28
29      if (!response.ok) {
30        throw new Error('Network response was not ok');
31      }
32
33      const newBook = await response.json();
34      onAddBook(newBook);
35      navigate('/');
36    } catch (error) {
37      setError('Failed to create book. Please try again.');
38      console.error('Error:', error);
39    }
40  };
}
```

❖ CreateBook.jsx X JS App.js

hw4-template > src > components > Create > ❖ CreateBook.jsx > ...

```
42 return (
43   <div className="create-book-container">
44     <h2>Add New Book</h2>
45     {error && <div className="error-message">{error}</div>}
46     <form onSubmit={handleSubmit}>
47       <div className="form-group">
48         <label htmlFor="title">Book Title:</label>
49         <input
50           type="text"
51           id="title"
52           value={title}
53           onChange={(e) => setTitle(e.target.value)}
54           required
55         />
56       </div>
57       <div className="form-group">
58         <label htmlFor="author">Author Name:</label>
59         <input
60           type="text"
61           id="author"
62           value={author}
63           onChange={(e) => setAuthor(e.target.value)}
64           required
65         />
66       </div>
67       <div className="form-group">
68         <label htmlFor="isbn">ISBN:</label>
69         <input
70           type="text"
71           id="isbn"
72           value={isbn}
73           onChange={(e) => setIsbn(e.target.value)}
74         />
75       </div>
76       <button type="submit" className="submit-button">Add Book</button>
77     </form>
78   </div>
79 );
80 };
81 
```

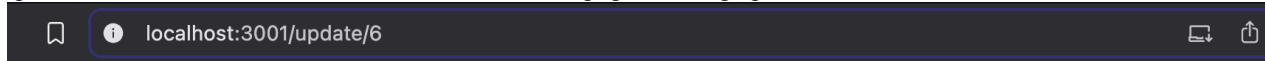
III. Update Book (2 Points)

Write the necessary code to create the 'UpdateBook.jsx' component. The component should be rendered when the user is on the '/update' route.

The component should accept props (similar to demo) to update the new book and have the following:

- An input field for Book Title.
- An input field for Author Name.
- A submit button labeled "Update Book".

When the user clicks the Update Book button the book within context should be updated and the user should be redirected to the home page showing updated book list.



Update Book

Book Title:

Test Book

Author Name:

Test Author

Update Book

Cancel



Update Book

Book Title:

Updated Test Book

Author Name:

Updated|Test Author

Update Book

Cancel



Book Management System

Add New Book

To Kill a Mockingbird

Author: Harper Lee

ISBN: 978-0446310789

Update

Delete

Atomic Habits

Author: James Clear

ISBN: 1234567

Update

Delete

Updated Test Book

Author: Updated Test Author

ISBN: 987654

Update

Delete

⌘ UpdateBook.jsx ×

```
hw4-template > src > components > Update > ⚡ UpdateBook.jsx > [?] UpdateBook > [?] fetchBook
 1 import React, { useState, useEffect } from 'react';
 2 import { useNavigate, useParams } from 'react-router-dom';
 3 import './UpdateBook.css';
 4
 5 const UpdateBook = ({ onUpdateBook }) => {
 6   const [title, setTitle] = useState('');
 7   const [author, setAuthor] = useState('');
 8   const [loading, setLoading] = useState(true);
 9   const [error, setError] = useState(null);
10   const navigate = useNavigate();
11   const { id } = useParams();
12
13   useEffect(() => {
14     fetchBook();
15   }, [id]);
16
17   const fetchBook = async () => {
18     try {
19       setLoading(true);
20       const response = await fetch(`http://localhost:3000/api/books/${id}`);
21       if (response.ok) {
22         const book = await response.json();
23         setTitle(book.title);
24         setAuthor(book.author);
25       } else {
26         throw new Error('Book not found');
27       }
28     } catch (error) {
29       setError(error.message);
30       console.error('Error fetching book:', error);
31     } finally {
32       setLoading(false);
33     }
34   };
35 }
```

⌘ UpdateBook.jsx ✎

```
hw4-template > src > components > Update > ⌘ UpdateBook.jsx > [o] UpdateBook > [o] fetchBook
  5  const UpdateBook = ({ onUpdateBook }) => {
  6    const handleSubmit = async (e) => {
  7      e.preventDefault();
  8      try {
  9        const response = await fetch(`http://localhost:3000/api/books/${id}`, {
 10          method: 'PUT',
 11          headers: {
 12            'Content-Type': 'application/json',
 13          },
 14          body: JSON.stringify({ title, author }),
 15        });
 16
 17        if (response.ok) {
 18          const updatedBook = await response.json();
 19          onUpdateBook && onUpdateBook(updatedBook);
 20          navigate('/');
 21        } else {
 22          throw new Error('Failed to update book');
 23        }
 24      } catch (error) {
 25        setError(error.message);
 26        console.error('Error:', error);
 27      }
 28    };
 29
 30    if (loading) return <div>Loading...</div>;
 31    if (error) return <div className="error-message">{error}</div>;
 32  
```

```
UpdateBook.jsx U X
hw4-template > src > components > Update > UpdateBook.jsx > UpdateBook > fetchBook
5  const UpdateBook = ({ onUpdateBook }) => {
63    return (
64      <div className="update-book-container">
65        <h2>Update Book</h2>
66        <form onSubmit={handleSubmit}>
67          <div className="form-group">
68            <label htmlFor="title">Book Title:</label>
69            <input
70              type="text"
71              id="title"
72              value={title}
73              onChange={(e) => setTitle(e.target.value)}
74              required
75            />
76          </div>
77          <div className="form-group">
78            <label htmlFor="author">Author Name:</label>
79            <input
80              type="text"
81              id="author"
82              value={author}
83              onChange={(e) => setAuthor(e.target.value)}
84              required
85            />
86          </div>
87          <div className="button-group">
88            <button type="submit" className="submit-button">Update Book</button>
89            <button type="button" onClick={() => navigate('/')} className="cancel-button">
90              Cancel
91            </button>
92          </div>
93        </form>
94      </div>
95    );
96  };
97
98  export default UpdateBook;
```

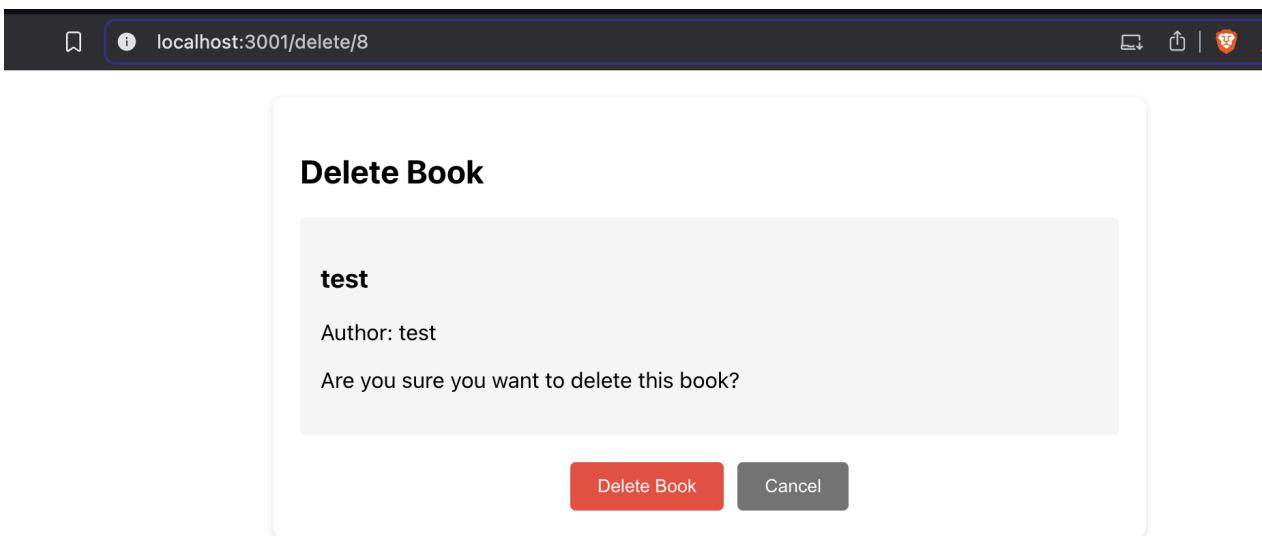
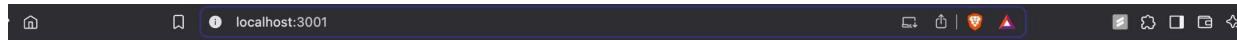
IV . Delete the Book (2 Points)

Write the necessary code to create the ‘DeleteBook.jsx’ component. The component should be rendered when the user is on the ‘/delete’ route.

The component should accept props (similar to demo) to delete a book and have the following:

- A button labeled "Delete Book".

When the user clicks the Delete Book button the book in context should be deleted and the user should be redirected to the home page showing the updated book list.



Book Management System

Add New Book

| | | | |
|---|---|--|--|
| To Kill a Mockingbird Author: Harper Lee ISBN: 978-0446310789 Update Delete | Atomic Habits Author: James Clear ISBN: 1234567 Update Delete | Updated Test Book Author: Updated Test Author ISBN: 987654 Update Delete | test Author: test ISBN: 1234 Update Delete |
|---|---|--|--|

hw4-template > src > components > Delete > *DeleteBook.jsx*

```
1 import React, { useState, useEffect } from 'react';
2 import { useNavigate, useParams } from 'react-router-dom';
3 import './DeleteBook.css';
4
5 const DeleteBook = ({ onDelete }) => {
6   const [book, setBook] = useState(null);
7   const [loading, setLoading] = useState(true);
8   const [error, setError] = useState(null);
9   const navigate = useNavigate();
10  const { id } = useParams();
11
12  useEffect(() => {
13    fetchBookDetails();
14  }, [id]);
15
16  const fetchBookDetails = async () => {
17    try {
18      const response = await fetch(`http://localhost:3000/api/books/${id}`);
19      if (response.ok) {
20        const data = await response.json();
21        setBook(data);
22      } else {
23        throw new Error('Book not found');
24      }
25    } catch (error) {
26      setError(error.message);
27      console.error('Error fetching book:', error);
28    } finally {
29      setLoading(false);
30    }
31  };
32
33  const handleDelete = async () => {
34    try {
35      const response = await fetch(`http://localhost:3000/api/books/${id}`, {
36        method: 'DELETE',
37      });
38    } catch (error) {
39      setError(error.message);
40      console.error('Error deleting book:', error);
41    }
42  };
43
44  const handleCancel = () => {
45    onDelete();
46  };
47
48  return (
49    <div>
50      <h2>Delete Book</h2>
51      <p>Are you sure you want to delete this book?</p>
52      <button onClick={handleDelete}>Delete</button>
53      <button onClick={handleCancel}>Cancel</button>
54    </div>
55  );
56}
```

```
hw4-template > src > components > Delete > DeleteBook.jsx

39  if (response.ok) {
40    onDelete && onDelete(id);
41    navigate('/');
42  } else {
43    throw new Error('Failed to delete book');
44  }
45  } catch (error) {
46    setError(error.message);
47    console.error('Error:', error);
48  }
49};
50
51 if (loading) return <div>Loading...</div>;
52 if (error) return <div className="error-message">{error}</div>;
53
54 return (
55   <div className="delete-book-container">
56     <h2>Delete Book</h2>
57     {book && (
58       <div className="book-details">
59         <h3>{book.title}</h3>
60         <p>Author: {book.author}</p>
61         <p className="warning-text">Are you sure you want to delete this book?</p>
62       </div>
63     )}
64     <div className="button-group">
65       <button onClick={handleDelete} className="delete-button">Delete Book</button>
66       <button onClick={() => navigate('/')} className="cancel-button">Cancel</button>
67     </div>
68   </div>
69 );
70 };
71
72 export default DeleteBook;
```

V . For all the above questions make sure to:

- Pass props for the Create, Update, and Delete components. **(1 Point)**

```

<Router>
  <div className="App">
    <Routes>
      <Route path="/" element={<Home books={books} onError={setError} />} />
      <Route path="/create" element={<CreateBook onAddBook={setBooks} />} />
      <Route
        path="/update/:id"
        element={<UpdateBook books={books} setBooks={setBooks} />}
      />
      <Route
        path="/delete/:id"
        element={<DeleteBook books={books} setBooks={setBooks} />}
      />
    </Routes>
  </div>
</Router>

```

- Use hooks like useState and useEffect wherever necessary. **(1 Point)**

```

const App = () => {
  const [books, setBooks] = useState([]);
  const [loading, setLoading] = useState(true);
  const [error, setError] = useState(null);

  useEffect(() => {
    fetchBooks();
  }, []);

  const fetchBooks = async () => {
    try {
      setLoading(true);
      const response = await fetch('http://localhost:3000/api/books');
      if (!response.ok) {
        throw new Error(`HTTP error! status: ${response.status}`);
      }
      const data = await response.json();
      setBooks(data);
    } catch (error) {
      setError('Failed to fetch books. Please ensure the server is running.');
      console.error('Error:', error);
    } finally {
      setLoading(false);
    }
  };
};

```

```
const CreateBook = ({ onAddBook }) => {
  const [title, setTitle] = useState('');
  const [author, setAuthor] = useState('');
  const [isbn, setIsbn] = useState('');
  const [error, setError] = useState(null);
  const navigate = useNavigate();
```

```
const UpdateBook = ({ onUpdateBook }) => [
  const [title, setTitle] = useState('');
  const [author, setAuthor] = useState('');
  const [loading, setLoading] = useState(true);
  const [error, setError] = useState(null);
  const navigate = useNavigate();
  const { id } = useParams();

  useEffect(() => {
    fetchBook();
  }, [id]);
```

```
const DeleteBook = ({ onDelete }) => {
  const [book, setBook] = useState(null);
  const [loading, setLoading] = useState(true);
  const [error, setError] = useState(null);
  const navigate = useNavigate();
  const { id } = useParams();

  useEffect(() => {
    fetchBookDetails();
  }, [id]);
```

- User React-Router-Dom for routing. (1 Point)

```
⚙️ Home.jsx ✘

hw4-template > src > components > Home > ⚙️ Home.jsx

1  import React, { useState, useEffect } from 'react';
2  import { Link } from 'react-router-dom';
3  import './Home.css';
4
5  const Home = () => [
6    const [books, setBooks] = useState([]);
7
8    useEffect(() => {
9      // Fetch books from API
10     fetch('http://localhost:3000/api/books')
11       .then(response => response.json())
12       .then(data => setBooks(data))
13       .catch(error => console.error('Error:', error));
14    }, []);
15
16    return (
17      <div className="home-container">
18        <h1>Book Management System</h1>
19        <Link to="/create" className="add-button">Add New Book</Link>
20
21        <div className="books-grid">
22          {books.map(book => (
23            <div key={book.id} className="book-card">
24              <h3>{book.title}</h3>
25              <p>Author: {book.author}</p>
26              <p>ISBN: {book.isbn}</p>
27              <div className="book-actions">
28                <Link to={`/update/${book.id}`} className="edit-button">Update</Link>
29                <Link to={`/delete/${book.id}`} className="delete-button">Delete</Link>
30              </div>
31            </div>
32          )));
33        </div>
34      </div>
35    );
36  ];
37
38  export default Home;
```

Q2. MySQL (10 points)

In this question, you will apply the concepts learned in class to build a simple CRUD (Create, Read, Update, Delete) application using Node.js and MySQL using Sequelize. Create an appropriate book Model, *View(Optional)*, and Controller.

You are required to create a Book Management System that allows users to:

- Add a new book (POST)

The screenshot shows the Postman interface with the following details:

- Collection:** Vimal Workspace
- Environment:** No environment
- Request Method:** POST
- Request URL:** http://localhost:3000/api/books
- Body (JSON):**

```
1 {
2   "title": "The Great Gatsby",
3   "author": "F. Scott Fitzgerald",
4   "isbn": "978-0743273565",
5   "publishedYear": 1925
6 }
7
```
- Response Status:** 201 Created
- Response Body (JSON):**

```
1 {
2   "id": 2,
3   "title": "The Great Gatsby",
4   "author": "F. Scott Fitzgerald",
5   "isbn": "978-0743273565",
6   "publishedYear": 1925,
7   "updatedAt": "2025-09-24T08:02:52.368Z",
```

Vimal Workspace

New Import

POST Get data

HTTP My Collection / Get data

Save Share

POST Get data

POST Post data

Params Auth Headers (8) Body Scripts Settings

raw JSON

1 {
2 "title": "To Kill a Mockingbird",
3 "author": "Harper Lee",
4 "isbn": "978-0446310789",
5 "publishedYear": 1960
6 }
7

Body 201 Created 10 ms 458 B Save Response

{ } JSON Preview Visualize

1 {
2 "id": 5,
3 "title": "To Kill a Mockingbird",
4 "author": "Harper Lee",
5 "isbn": "978-0446310789",
6 "publishedYear": 1960,
7 "updatedAt": "2025-09-24T08:06:21.293Z",
8 "createdAt": "2025-09-24T08:06:21.293Z"
9 }

- View all books (GET)

The screenshot shows the Postman application interface. On the left, there's a sidebar with 'Vimal Workspace' at the top, followed by 'Collections', 'Environments', 'Flows', and 'History'. The main area has tabs for 'Get data' (which is selected), 'Overview', and '+'. Below the tabs, it says 'HTTP My Collection / Get data' and shows a 'Send' button. Underneath, there are sections for 'Params', 'Auth', 'Headers (8)', 'Body', 'Scripts', 'Settings', and 'Cookies'. The 'Body' section is expanded, showing a table for 'Query Params' with columns for 'Key' and 'Value'. The 'Body' tab also shows a preview of the response. The response itself is a JSON array with two elements, each representing a book:

```

1  [
2   {
3     "id": 2,
4     "title": "The Great Gatsby",
5     "author": "F. Scott Fitzgerald",
6     "isbn": "978-0743273565",
7     "publishedYear": 1925,
8     "createdAt": "2025-09-24T08:02:52.000Z",
9     "updatedAt": "2025-09-24T08:02:52.000Z"
10    },
11    {
12      "id": 5,
13      "title": "To Kill a Mockingbird",
14      "author": "Harper Lee",
15      "isbn": "978-0446310789",
16      "publishedYear": 1960,
17      "createdAt": "2025-09-24T08:06:21.000Z",
18      "updatedAt": "2025-09-24T08:06:21.000Z"
19    }
20  ]

```

- View a book by ID (GET)

The screenshot shows the Postman application interface. On the left, the sidebar has sections for Collections, Environments, Flows, and History. The main area shows a collection named "My Collection" with two items: "POST Get data" and "POST Post data". A specific POST request titled "Get data" is selected, which is a GET request to the URL `http://localhost:3000/api/books/2`. The "Params" tab is active, showing a table with one row and columns for Key, Value, Description, and Bulk Edit. Below the table, the "Body" tab displays the response in JSON format:

```
1 {  
2   "id": 2,  
3   "title": "The Great Gatsby",  
4   "author": "F. Scott Fitzgerald",  
5   "isbn": "978-0743273565",  
6   "publishedYear": 1925,  
7   "createdAt": "2025-09-24T08:02:52.000Z",  
8   "updatedAt": "2025-09-24T08:02:52.000Z"  
9 }
```

- Update book details (PUT)

The screenshot shows the Postman application interface. On the left, the sidebar includes 'Vimal Workspace' with 'Collections', 'Environments', 'Flows', and 'History'. The main workspace shows a collection named 'My Collection' with 'POST Get data' and 'POST Post data' operations. A 'PUT Get data' operation is selected. The 'Body' tab is active, displaying the following JSON payload:

```
1 {
2   "title": "Updated Title",
3   "author": "Updated Author",
4   "isbn": "978-0743273565",
5   "publishedYear": 2024
6 }
7
```

Below the body, the response status is '200 OK' with a response time of '39 ms' and a size of '449 B'. The response body is also shown in JSON format:

```
1 {
2   "id": 2,
3   "title": "Updated Title",
4   "author": "Updated Author",
5   "isbn": "978-0743273565",
6   "publishedYear": 2024,
7   "createdAt": "2025-09-24T08:02:52.000Z",
8   "updatedAt": "2025-09-24T08:09:06.280Z"
9 }
```

Vimal Workspace

New Import

Get data Overview No environment

Collections + Search collections

My Collection

POST Get data

POST Post data

HTTP My Collection / Get data

Save Share

Send

GET http://localhost:3000/api/books/2

Params Auth Headers (8) Body Scripts Settings Cookies

Query Params

| Key | Value | Description | Bulk Edit |
|-----|-------|-------------|-----------|
| Key | Value | Description | |

Body 200 OK 39 ms 449 B Save Response

[{} JSON ▾] Preview Visualize

```
1 {  
2   "id": 2,  
3   "title": "Updated Title",  
4   "author": "Updated Author",  
5   "isbn": "978-0743273565",  
6   "publishedYear": 2024,  
7   "createdAt": "2025-09-24T08:02:52.000Z",  
8   "updatedAt": "2025-09-24T08:09:06.280Z"  
9 }
```

- Delete a book (DELETE)

The screenshot shows the Postman application interface. On the left, there's a sidebar with 'Vimal Workspace' containing sections for 'Collections', 'Environments', 'Flows', and 'History'. The main area has tabs for 'New' and 'Import'. A 'DEL Get data' tab is active, showing a collection named 'My Collection' with a 'Get data' operation. The main workspace displays a DELETE request to 'http://localhost:3000/api/books/2'. The 'Headers' tab shows '(8)' headers. The 'Body' tab is set to 'JSON' and contains the value '{ }'. The response section shows a status of '204 No Content' with a duration of '27 ms' and a size of '166 B'. The response body is empty, indicated by the number '1'.

Vimal Workspace

New Import

GET Get data Overview + No environment

HTTP My Collection / Get data

Save Share

POST Get data

POST Post data

Params Auth Headers (8) Body Scripts Settings Cookies

raw JSON

1 {
2 "title": "Updated Title",
3 "author": "Updated Author",
4 "isbn": "978-0743273565",
5 "publishedYear": 2024
6 }
7

Body 200 OK 21 ms 455 B Save Response

{ } JSON Preview Visualize

1 [
2 {
3 "id": 5,
4 "title": "To Kill a Mockingbird",
5 "author": "Harper Lee",
6 "isbn": "978-0446310789",
7 "publishedYear": 1960,
8 "createdAt": "2025-09-24T08:06:21.000Z",
9 "updatedAt": "2025-09-24T08:06:21.000Z"
10 }
11]

Create a MySQL database named **book_db** with a table called **books** for this task.

```
[mysql]> CREATE DATABASE book_db;
Query OK, 1 row affected (0.01 sec)

[mysql]> show databases
[    -> ;
+-----+
| Database      |
+-----+
| book_db       |
| information_schema |
| mysql          |
| performance_schema |
| sys            |
| ubereats       |
+-----+
6 rows in set (0.01 sec)
```

```
mysql> use book_db;
[Reading table information for completion of table and column names
You can turn off this feature to get a quicker startup with -A
[
Database changed
mysql> show tables;
+-----+
| Tables_in_book_db |
+-----+
| Books             |
+-----+
1 row in set (0.01 sec)

mysql> describe books1
-> ;
ERROR 1146 (42S02): Table 'book_db.books1' doesn't exist
mysql> describe books;
+-----+-----+-----+-----+-----+-----+
| Field      | Type       | Null | Key | Default | Extra        |
+-----+-----+-----+-----+-----+-----+
| id         | int        | NO   | PRI | NULL    | auto_increment
| title      | varchar(255) | NO   |     | NULL    |
| author     | varchar(255) | NO   |     | NULL    |
| isbn       | varchar(255) | YES  | UNI | NULL    |
| publishedYear | int        | YES  |     | NULL    |
| createdAt  | datetime   | NO   |     | NULL    |
| updatedAt  | datetime   | NO   |     | NULL    |
+-----+-----+-----+-----+-----+-----+
7 rows in set (0.01 sec)
```

```
mysql-nodejs > JS book.model.js > ...
1   const { DataTypes } = require('sequelize');
2   const sequelize = require('../config/database');
3
4   const Book = sequelize.define('Book', {
5     title: {
6       type: DataTypes.STRING,
7       allowNull: false
8     },
9     author: {
10       type: DataTypes.STRING,
11       allowNull: false
12     },
13     isbn: {
14       type: DataTypes.STRING,
15       unique: true
16     },
17     publishedYear: {
18       type: DataTypes.INTEGER
19     }
20   });
21
22   module.exports = Book;
```

You are free to name your API endpoints appropriately.



DATA 236 - HW 4

POST `http://localhost:3000/api/books`

GET `http://localhost:3000/api/books`

GET `http://localhost:3000/api/books/:id`

PUT `http://localhost:3000/api/books/:id`

DEL `http://localhost:3000/api/books/:id`

Submit the Postman API response screenshots for each operation. Each completed operation will get 2 points. Also, submit a screenshot of the database and project folder structure.

```
✓ HW 4
  ✓ hw4-template
    > node_modules
    > public
    ✓ src
      > components
      # App.css
      JS App.js
      # index.css
      JS index.js
      JS setupTests.js
      ♦ .gitignore
      {} package-lock.json
      {} package.json
      ⓘ README.md
```

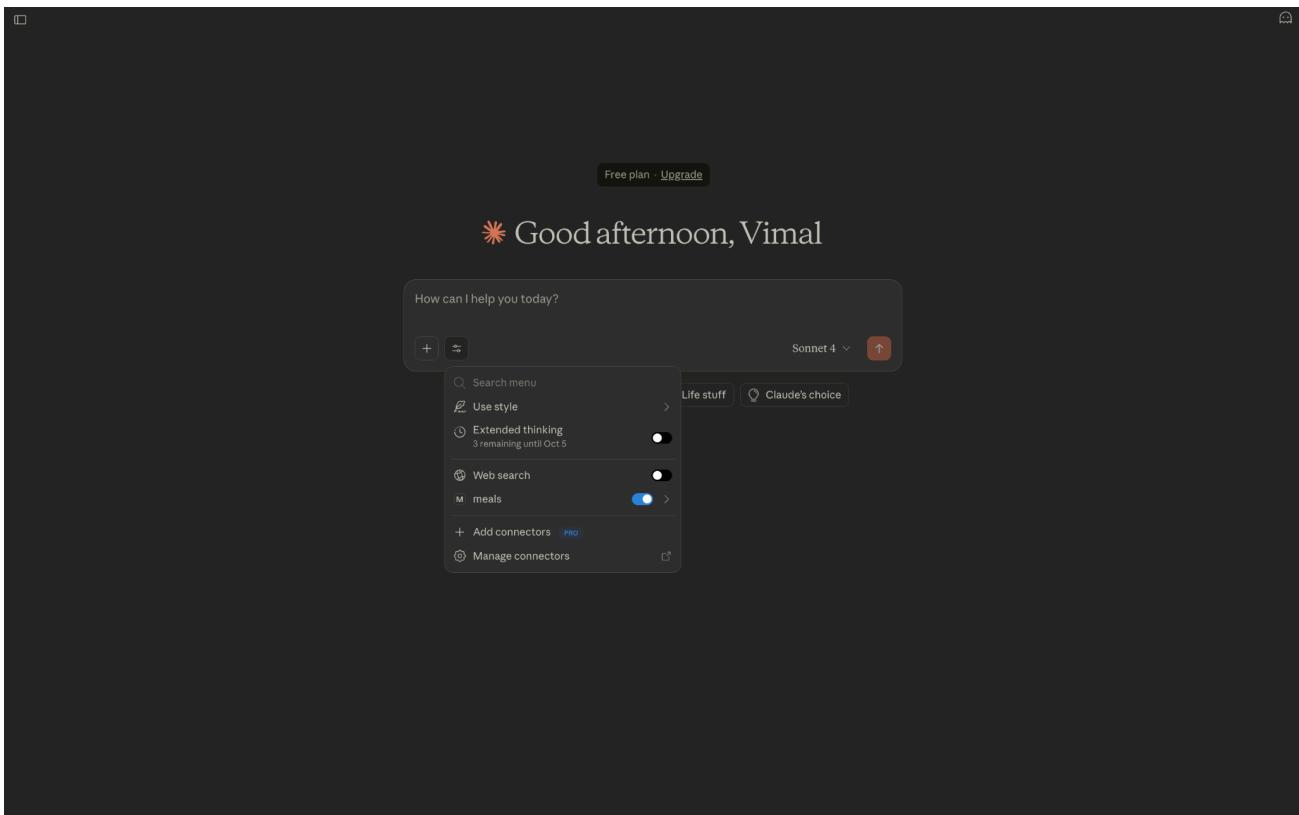
```
▽ HW 4
  > hw4-template
  ▽ mysql-nodejs
    > node_modules
    ▽ src
      > config
      > controllers
      > models
      > routes
      JS app.js
      JS user.js
      ⚙ .env
      {} package-lock.json
      {} package.json
      ♦ .gitignore
```

```
mysql-nodejs > src > JS app.js > ⌂ startServer
 1  const express = require('express');
 2  const cors = require('cors');
 3  const sequelize = require('./config/database');
 4  const bookRoutes = require('./routes/book.routes');
 5
 6  const app = express();
 7
 8  // Middleware
 9  app.use(cors());
10  app.use(express.json());
11
12 // Routes
13 app.use('/api', bookRoutes);
14
15 // Database connection and server start
16 const PORT = process.env.PORT || 3000;
17
18 async function startServer() {
19   try {
20     await sequelize.authenticate();
21     console.log('Database connected successfully');
22     await sequelize.sync();
23
24     app.listen(PORT, () => {
25       console.log(`Server running on port ${PORT}`);
26     });
27   } catch (error) {
28     console.error('Unable to start server:', error);
29   }
30 }
31
32 startServer();
33
34 module.exports = app;
```

Homework: Build a MCP server that fetches recipes from TheMealDB (10 points)

What to show (prompts to try in Claude)

Once your server is connected and visible in Claude:



- “Find 3 Italian pasta recipes by name and show short summaries with images.”

Classic Italian pasta recipes

v Find 3 Italian pasta recipes by name and show short summaries with images.

I'll help you find 3 Italian pasta recipes! Let me search for some popular Italian pasta dishes.

Search meals by name

Let me search for more specific Italian pasta dishes:

Search meals by name

Search meals by name

Perfect! Now let me get the detailed information for these three authentic Italian pasta recipes:

Meal details

Meal details

Meal details

Here are 3 delicious Italian pasta recipes with short summaries and images:

1. Spaghetti Bolognese

Show Image

Summary: A classic Italian comfort dish featuring al dente spaghetti topped with a rich, hearty meat sauce made from ground beef, onions, garlic, mushrooms, and tomatoes. The sauce is slowly simmered with herbs and beef stock to develop deep, savory flavors. Traditionally served with freshly grated Parmesan cheese.

Key ingredients: Spaghetti, lean minced beef, onions, garlic, mushrooms, tomatoes, oregano

2. Spaghetti alla Carbonara

Show Image

Reply to Claude...

Sonnet 4

Summary: A classic Italian comfort dish featuring al dente spaghetti topped with a rich, hearty meat sauce made from ground beef, onions, garlic, mushrooms, and tomatoes. The sauce is slowly simmered with herbs and beef stock to develop deep, savory flavors. Traditionally served with freshly grated Parmesan cheese.

Key ingredients: Spaghetti, lean minced beef, onions, garlic, mushrooms, tomatoes, oregano

2. Spaghetti alla Carbonara

Show Image

Summary: An authentic Roman pasta dish that creates a silky, creamy sauce without using cream - just eggs, cheese, and pasta water. The hot pasta is tossed with crispy pancetta (or bacon), beaten eggs, pecorino romano, and black pepper to create a luxurious coating. The key is timing and temperature to avoid scrambling the eggs.

Key ingredients: Spaghetti, pancetta/bacon, eggs, pecorino cheese, black pepper

Classic Italian pasta recipes

3. Spicy Arrabiata Penne

Show Image

Summary: A fiery vegetarian pasta from Rome meaning “angry” in Italian, referring to the spicy red chili flakes. This simple yet flavorful dish combines penne pasta with a zesty tomato sauce infused with garlic, olive oil, and red pepper flakes. Fresh basil and Parmigiano-Reggiano finish this vibrant dish.

Key ingredients: Penne pasta, tomatoes, garlic, red chili flakes, basil, olive oil

All three recipes represent authentic Italian pasta preparations with distinct regional characteristics and cooking techniques!

Retry

Claude can make mistakes. Please double-check responses.

Reply to Claude...

Sonnet 4

- “List meals that use **chicken** as a main ingredient; include meal IDs so I can ask for details.”

The screenshot shows a dark-themed AI interface titled "Chicken meal recipe collection". At the top, there's a text input field containing a query: "List meals that use chicken as a main ingredient; include meal IDs so I can ask for details." Below the query, a message states: "I'll search for meals that use chicken as the main ingredient and provide you with the meal IDs for easy reference." A dropdown menu is open, showing the option "Meals by ingredient". The main content area displays a list of 12 chicken-based meals with their IDs:

- ID 52940 - Brown Stew Chicken ID 52846 - Chicken & mushroom Hotpot
- ID 52796 - Chicken Alfredo Primavera ID 52934 - Chicken Basquaise ID 52956 - Chicken Congee ID 52795 - Chicken Handi ID 52831 - Chicken Karaage ID 52813 - Kentucky Fried Chicken ID 52945 - Kung Pao Chicken ID 52774 - Pad See Ew ID 53039 - Piri-piri chicken and slaw ID 52814 - Thai Green Curry

Below this list, a message says: "You can ask me for detailed information about any of these meals by referencing their ID number. Just say something like "Give me details for meal ID 52945" and I'll provide you with the full recipe, ingredients, and cooking instructions!"

At the bottom, there's a text input field labeled "Reply to Claude...", a toolbar with icons for +, ≈, and a red button, and a status bar indicating "Sonnet 4" and a small upward arrow icon.

- “Show full details for meal id 52772 .”

Meal details lookup ▾

Show full details for meal id 52772

Meal details

Here are the full details for meal ID 52772:

Teriyaki Chicken Casserole

Category: Chicken
Cuisine: Japanese
Image:

Show Image

Ingredients:

- 3/4 cup soy sauce
- 1/2 cup water
- 1/4 cup brown sugar
- 1/2 teaspoon ground ginger
- 1/2 teaspoon minced garlic
- 4 tablespoons cornstarch
- 2 chicken breasts
- 1 (12 oz.) package stir-fry vegetables
- 3 cups brown rice

Instructions:

1. Preheat oven to 350°F. Spray a 9x13-inch baking pan with non-stick spray.
2. Combine soy sauce, 1/2 cup water, brown sugar, ginger and garlic in a small saucepan and cover. Bring to a boil over medium heat. Remove lid and cook for one minute once boiling.

Reply to Claude...

+

Sonnet 4 ▾ ↑

Meal details lookup ▾

- 4 tablespoons cornstarch
 - 2 chicken breasts
 - 1 (12 oz.) package stir-fry vegetables
 - 3 cups brown rice

Instructions:

1. Preheat oven to 350°F. Spray a 9x13-inch baking pan with non-stick spray.
 2. Combine soy sauce, $\frac{1}{2}$ cup water, brown sugar, ginger and garlic in a small saucepan and cover. Bring to a boil over medium heat. Remove lid and cook for one minute once boiling.
 3. Meanwhile, stir together the corn starch and 2 tablespoons of water in a separate dish until smooth. Once sauce is boiling, add mixture to the saucepan and stir to combine. Cook until the sauce starts to thicken then remove from heat.
 4. Place the chicken breasts in the prepared pan. Pour one cup of the sauce over top of chicken. Place chicken in oven and bake 35 minutes or until cooked through. Remove from oven and shred chicken in the dish using two forks.
 5. Meanwhile, steam or cook the vegetables according to package directions.
 6. Add the cooked vegetables and rice to the casserole dish with the chicken. Add most of the remaining sauce, reserving a bit to drizzle over the top when serving. Gently toss everything together in the casserole dish until combined. Return to oven and cook 15 minutes. Remove from oven and let stand 5 minutes before serving. Drizzle each serving with remaining sauce. Enjoy!

Video Tutorial:

[Watch on YouTube](#)

This is a complete one-dish meal that combines tender shredded chicken with a sweet and savory teriyaki sauce, vegetables, and brown rice all baked together in a casserole format.

 Claude does not have internet access enabled. Enable web search for up to date and accurate links.



 Retry ▾

Reply to Claude..

Sonnet 4 ✓

- “Give me one **random** meal with ingredients and measures.”

Random culinary creation

Give me one **random** meal with ingredients and measures.

M Random meal

Here's your random meal: **Fish Soup (Ukha)** - a traditional Russian seafood soup!

Ingredients & Measurements:

- Olive Oil - 2 tablespoons
- Onion - 1 sliced
- Carrots - 2 medium
- Fish Stock - 3 cups
- Water - 3 cups
- Potatoes - 4 large
- Bay Leaf - 3 leaves
- Cod - 1 piece
- Salmon - 1 piece

Instructions:

1. Heat olive oil in a medium pot over medium-high heat. Add onions and cook until they start to caramelize.
2. Add carrots and cook until they begin to soften (about 4 minutes).
3. Add stock, water, potatoes, bay leaves, and black peppercorns. Season with salt and bring to a boil.
4. Reduce heat, cover and cook for 10 minutes. Add millet and cook for 15 more minutes until millet and potatoes are cooked.
5. Gently add the fish cubes. Bring to a simmer - the fish cooks very quickly, so don't overcook. They're done when the flesh is opaque and flakes easily.
6. Garnish with chopped fresh dill or parsley before serving.

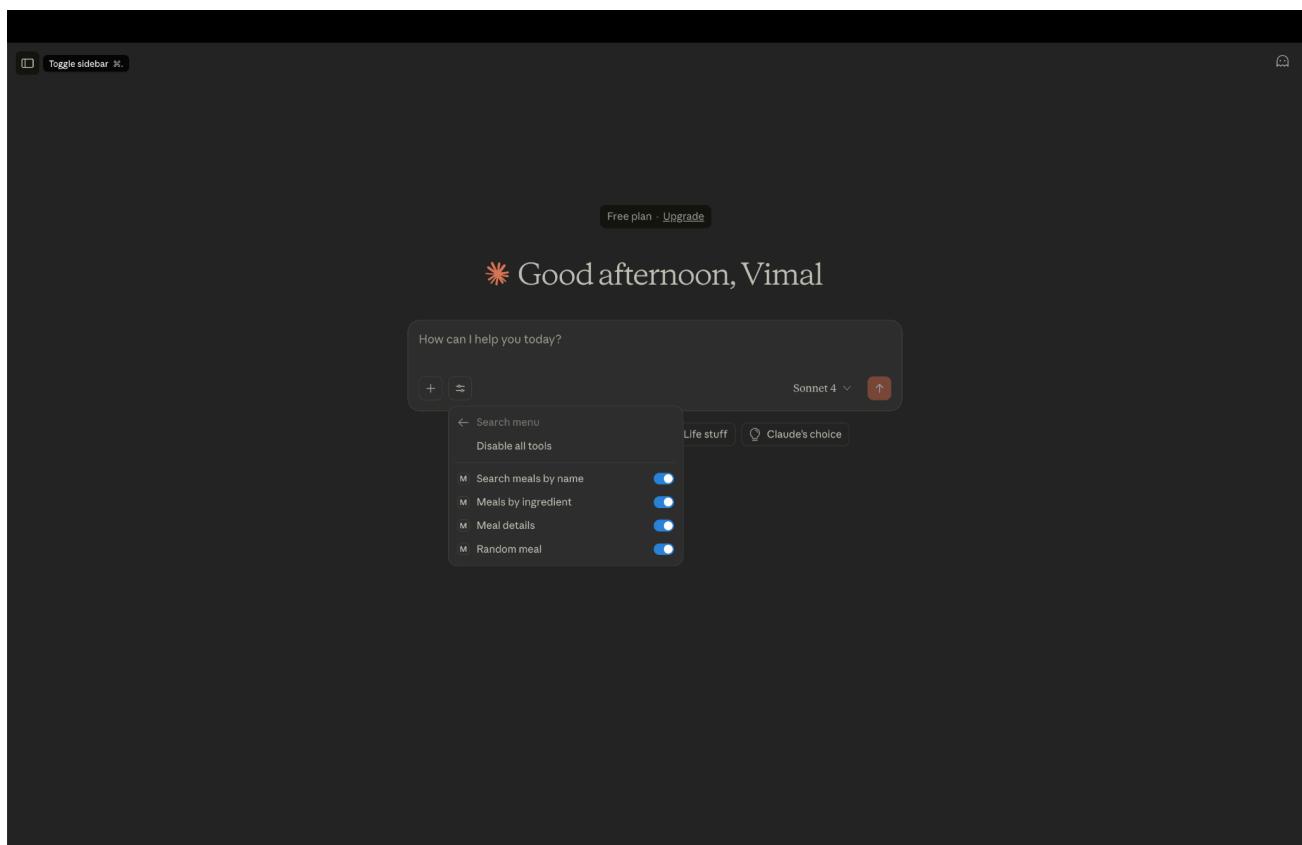
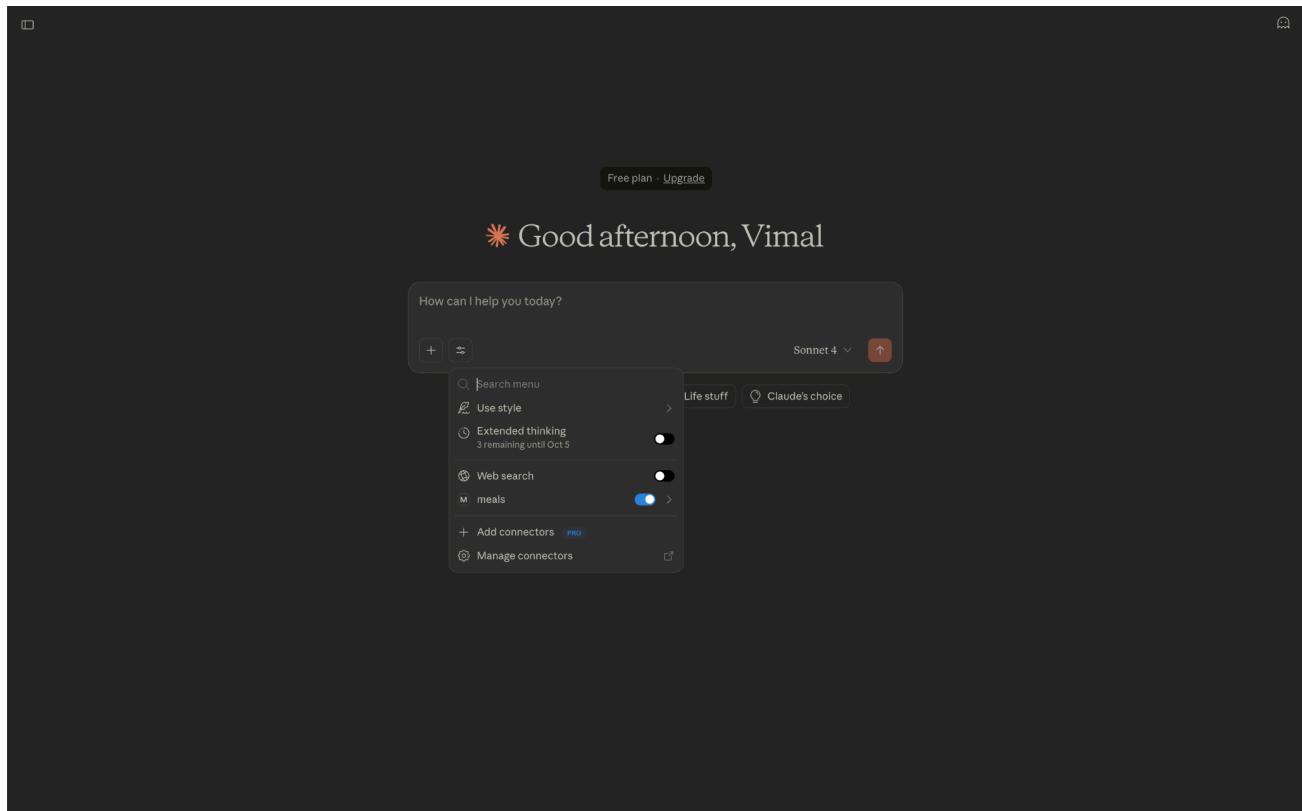
This hearty Russian soup is perfect for a comforting meal!

Reply to Claude...

Sonnet 4

1. Screenshot(s) and code snippet :

- Claude Desktop showing your server's tools and a sample query/answer.



FULL CODE:

```
.env ✘ meals_server_mcp.py 1, U ✘ {} claude_desktop_config.json
DATA-236---Distributed-Systems-for-Data-Engineering > Assignments > Assignment 4 > meals_server_mcp.py > MealDBClient > random
  1  from urllib import error, parse, request
  2  import json
  3  import logging
  4
  5  from mcp.server.fastmcp import FastMCP
  6
  7  LOGGER = logging.getLogger(__name__)
  8  if not LOGGER.handlers:
  9      handler = logging.StreamHandler()
 10      handler.setFormatter(logging.Formatter("%(asctime)s %(levelname)s %(message)s"))
 11      LOGGER.addHandler(handler)
 12  LOGGER.setLevel(logging.INFO)
 13
 14 API_ROOT = "https://www.themealdb.com/api/json/v1/1/"
 15 USER_AGENT = "MealHubMCP/1.0"
 16 DEFAULT_TIMEOUT = 15 # seconds
 17 SEARCH_LIMIT = 25
 18 FILTER_LIMIT = 50
 19 MAX_INGREDIENT_FIELDS = 20
 20
 21
 22 class MealServiceError(RuntimeError):
 23     pass
 24
 25
 26 class MealDBClient:
 27     def __init__(self, api_root=API_ROOT, timeout=DEFAULT_TIMEOUT):
 28         self.api_root = api_root if api_root.endswith("/") else f"{api_root}/"
 29         self.timeout = timeout
 30
 31     def _url(self, endpoint, params=None):
 32         params = params or {}
 33         query = parse.urlencode({k: v for k, v in params.items() if v not in (None, "")})
 34         url = parse.urljoin(self.api_root, endpoint)
 35         return f"{url}?{query}" if query else url
 36
 37     def _fetch(self, endpoint, params=None):
 38         url = self._url(endpoint, params)
 39         LOGGER.debug("Fetching %s", url)
 40         req = request.Request(url, headers={"User-Agent": USER_AGENT})
 41         try:
 42             with request.urlopen(req, timeout=self.timeout) as resp:
 43                 if resp.status != 200:
 44                     raise MealServiceError(f"TheMealDB returned HTTP {resp.status} for {url}")
 45                 try:
 46                     return json.load(resp)
 47                 except json.JSONDecodeError as exc:
 48                     raise MealServiceError("TheMealDB response was not valid JSON") from exc
 49             except error.URLError as exc:
 50                 raise MealServiceError(f"Problem reaching TheMealDB ({url}): {exc}") from exc
 51
 52     def search_by_name(self, query):
 53         payload = self._fetch("search.php", {"s": query})
 54         return payload.get("meals") or []
 55
 56     def filter_by_ingredient(self, ingredient):
 57         payload = self._fetch("filter.php", {"i": ingredient})
 58         return payload.get("meals") or []
 59
 60     def lookup(self, meal_id):
 61         payload = self._fetch("lookup.php", {"i": meal_id})
 62         return payload.get("meals") or []
 63
 64     def random(self):
 65         payload = self._fetch("random.php")
 66         return payload.get("meals") or []
```

```
⚙ .env ⚙  ⚙ meals_server_mcp.py 1, U  ⚙  {} claude_desktop_config.json
DATA-236---Distributed-Systems-for-Data-Engineering > Assignments > Assignment 4 > ⚙ meals_server_mcp.py > ⚙ meals_by_ingredient

67
68     def _summarize_search(meal):
69         return {
70             "id": meal.get("idMeal"),
71             "name": meal.get("strMeal"),
72             "area": meal.get("strArea"),
73             "category": meal.get("strCategory"),
74             "thumb": meal.get("strMealThumb"),
75         }
76
77
78     def _summarize_ingredient_card(meal):
79         return {
80             "id": meal.get("idMeal"),
81             "name": meal.get("strMeal"),
82             "thumb": meal.get("strMealThumb"),
83         }
84
85
86     def _gather_ingredients(meal):
87         items = []
88         for index in range(1, MAX_INGREDIENT_FIELDS + 1):
89             name = (meal.get(f"strIngredient{index}") or "").strip()
90             measure = (meal.get(f"strMeasure{index}") or "").strip()
91             if name:
92                 items.append({"name": name, "measure": measure or None})
93         return items
94
95
96     def _normalize_detail(meal):
97         return {
98             "id": meal.get("idMeal"),
99             "name": meal.get("strMeal"),
100            "category": meal.get("strCategory"),
101            "area": meal.get("strArea"),
102            "instructions": meal.get("strInstructions"),
103            "image": meal.get("strMealThumb"),
104            "source": meal.get("strSource"),
105            "youtube": meal.get("strYoutube"),
106            "ingredients": _gather_ingredients(meal),
107        }
108
109
110    client = MealDBClient()
111    mcp = FastMCP("meals")
112
113
114    def _clamp_limit(limit, *, ceiling):
115        if not isinstance(limit, int):
116            raise ValueError("limit must be an integer")
117        return max(1, min(limit, ceiling))
118
119
120    @mcp.tool()
121    def search_meals_by_name(query, limit=5):
122        query = query.strip()
123        if not query:
124            raise ValueError("query must not be empty")
125
126        limit = _clamp_limit(limit, ceiling=SEARCH_LIMIT)
127        meals = client.search_by_name(query)
128        if not meals:
129            return {"message": "no matches", "results": []}
130
131        return {"results": [_summarize_search(meal) for meal in meals[:limit]]}
132
133
```

```
⚙ .env • ⚡ meals_server_mcp.py 1, U • {} claude_desktop_config.json
DATA-236---Distributed-Systems-for-Data-Engineering > Assignments > Assignment 4 > ⚡ meals_server_mcp.py > ...
120     @mcp.tool()
121     def search_meals_by_name(query, limit=5):
122         query = query.strip()
123         if not query:
124             raise ValueError("query must not be empty")
125
126         limit = _clamp_limit(limit, ceiling=SEARCH_LIMIT)
127         meals = client.search_by_name(query)
128         if not meals:
129             return {"message": "no matches", "results": []}
130
131         return {"results": [_summarize_search(meal) for meal in meals[:limit]]}
132
133
134     @mcp.tool()
135     def meals_by_ingredient(ingredient, limit=12):
136         ingredient = ingredient.strip()
137         if not ingredient:
138             raise ValueError("ingredient must not be empty")
139
140         limit = _clamp_limit(limit, ceiling=FILTER_LIMIT)
141         meals = client.filter_by_ingredient(ingredient)
142         if not meals:
143             return {"message": "no matches", "results": []}
144
145         return {"results": [_summarize_ingredient_card(meal) for meal in meals[:limit]]}
146
147
148     @mcp.tool()
149     def meal_details(meal_id):
150         normalized_id = str(meal_id).strip()
151         if not normalized_id:
152             raise ValueError("id must not be empty")
153
154         meals = client.lookup(normalized_id)
155         if not meals:
156             return {"message": "no matches", "meal": None}
157
158         return {"meal": _normalize_detail(meals[0])}
159
160
161     @mcp.tool()
162     def random_meal():
163         meals = client.random()
164         if not meals:
165             return {"message": "no matches", "meal": None}
166         return {"meal": _normalize_detail(meals[0])}
167
168
169     if __name__ == "__main__":
170         mcp.run()
```