DATA-236 - Distributed Systems for Data Engineering

HOMEWORK 5

Vimalanandhan Sivanandham

017596436

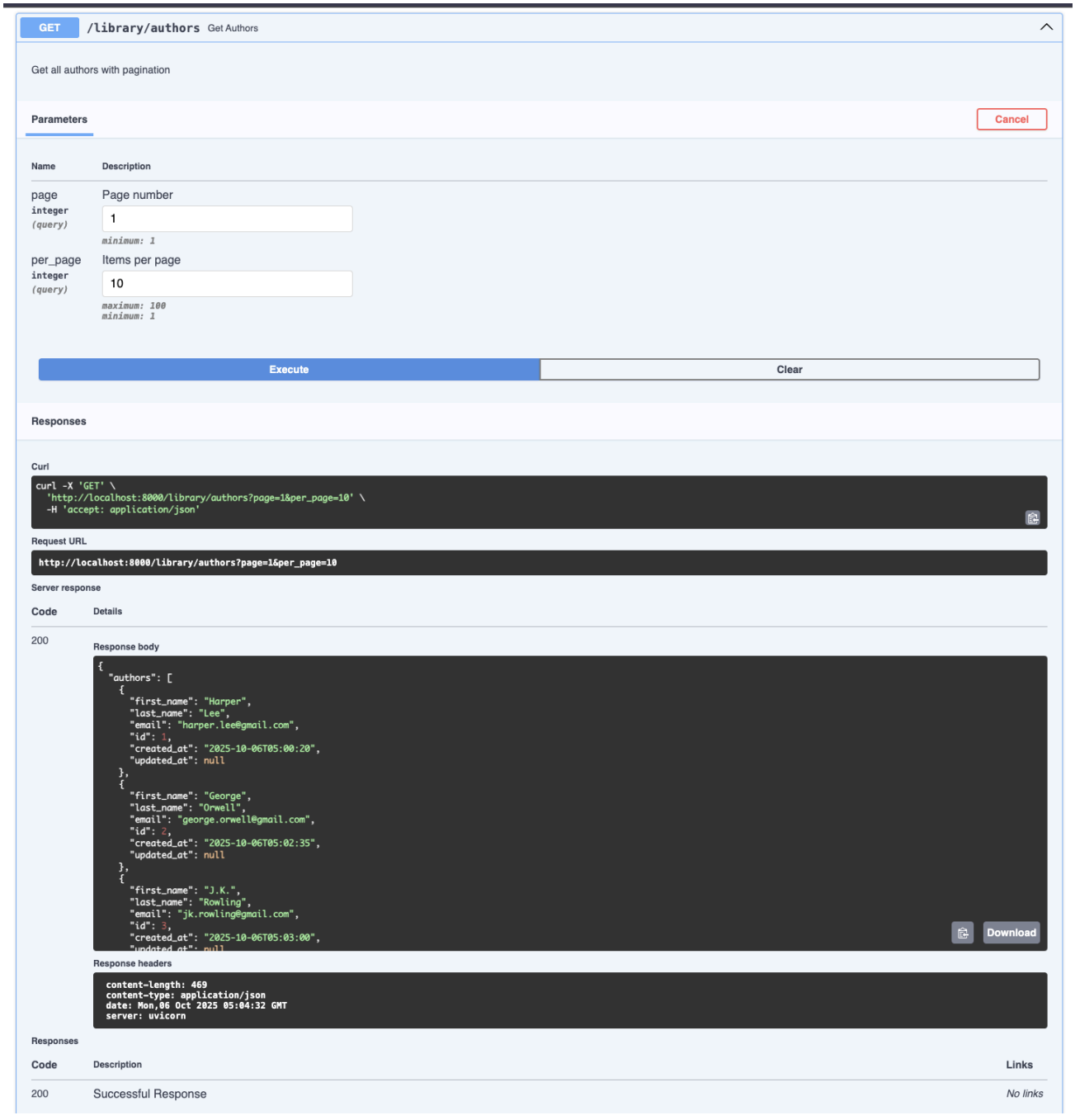
GitHub -

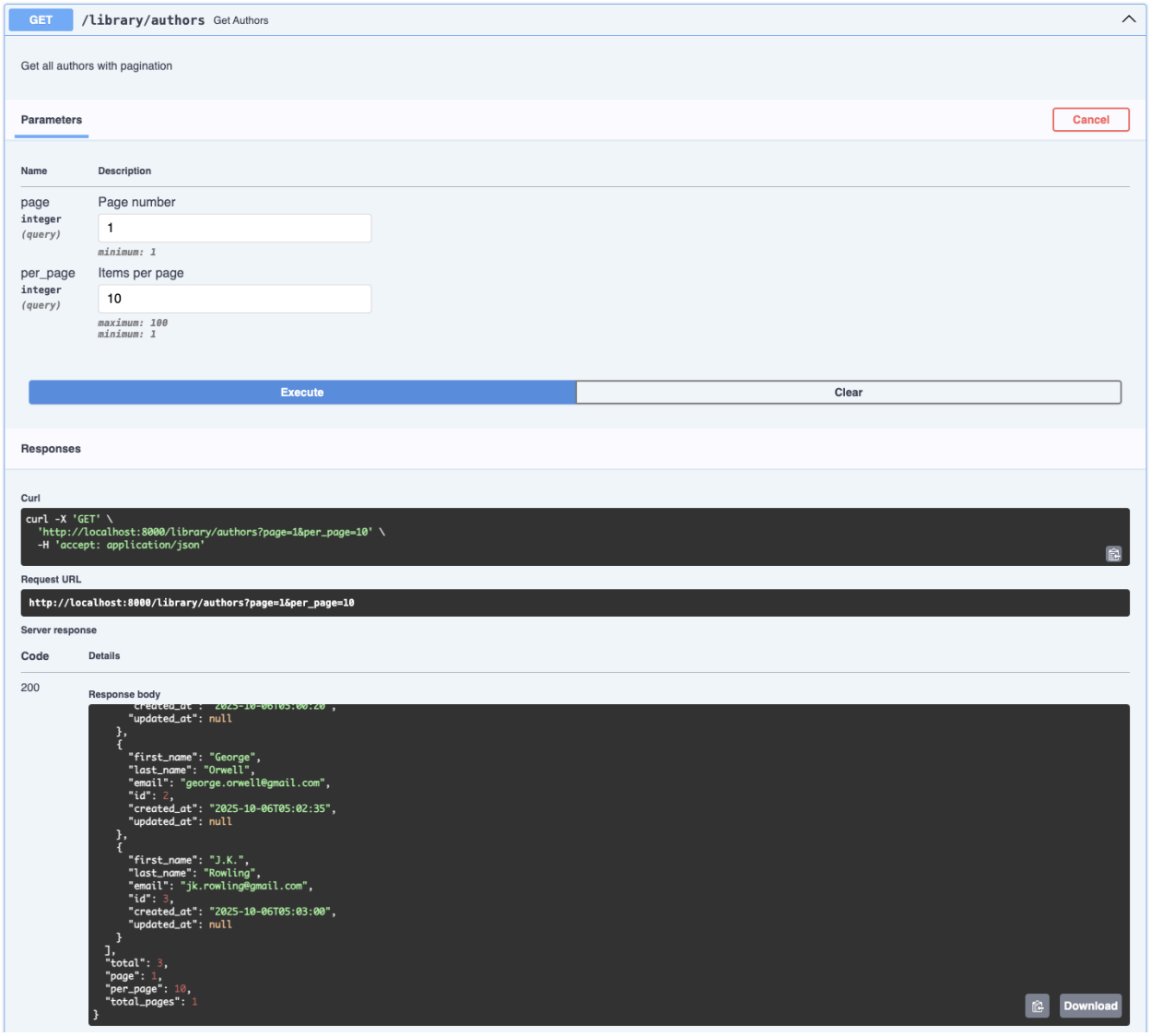
https://github.com/Vimalanandhan/DATA-236---Distributed-Systems-for-Data-Engineering/tree/main/Assignments/Assignment%205

PART -1

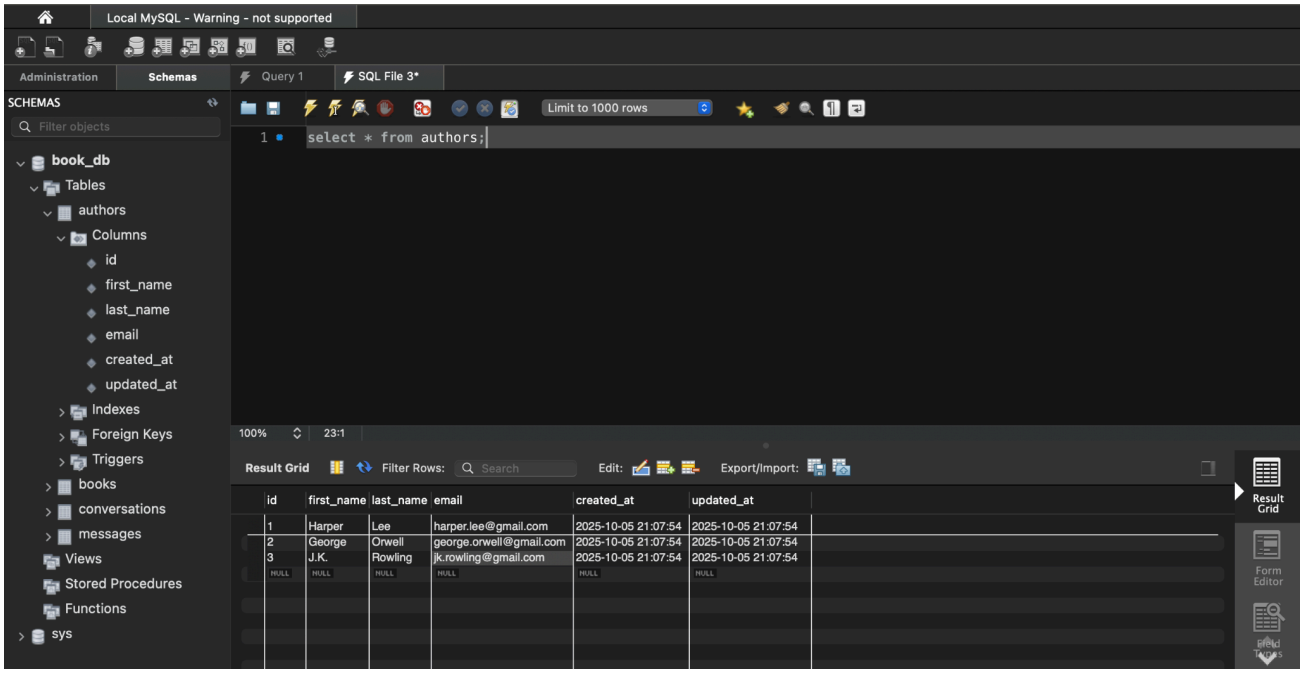
AUTHORS API endpoints

1. GET /library/authors - Get all authors.

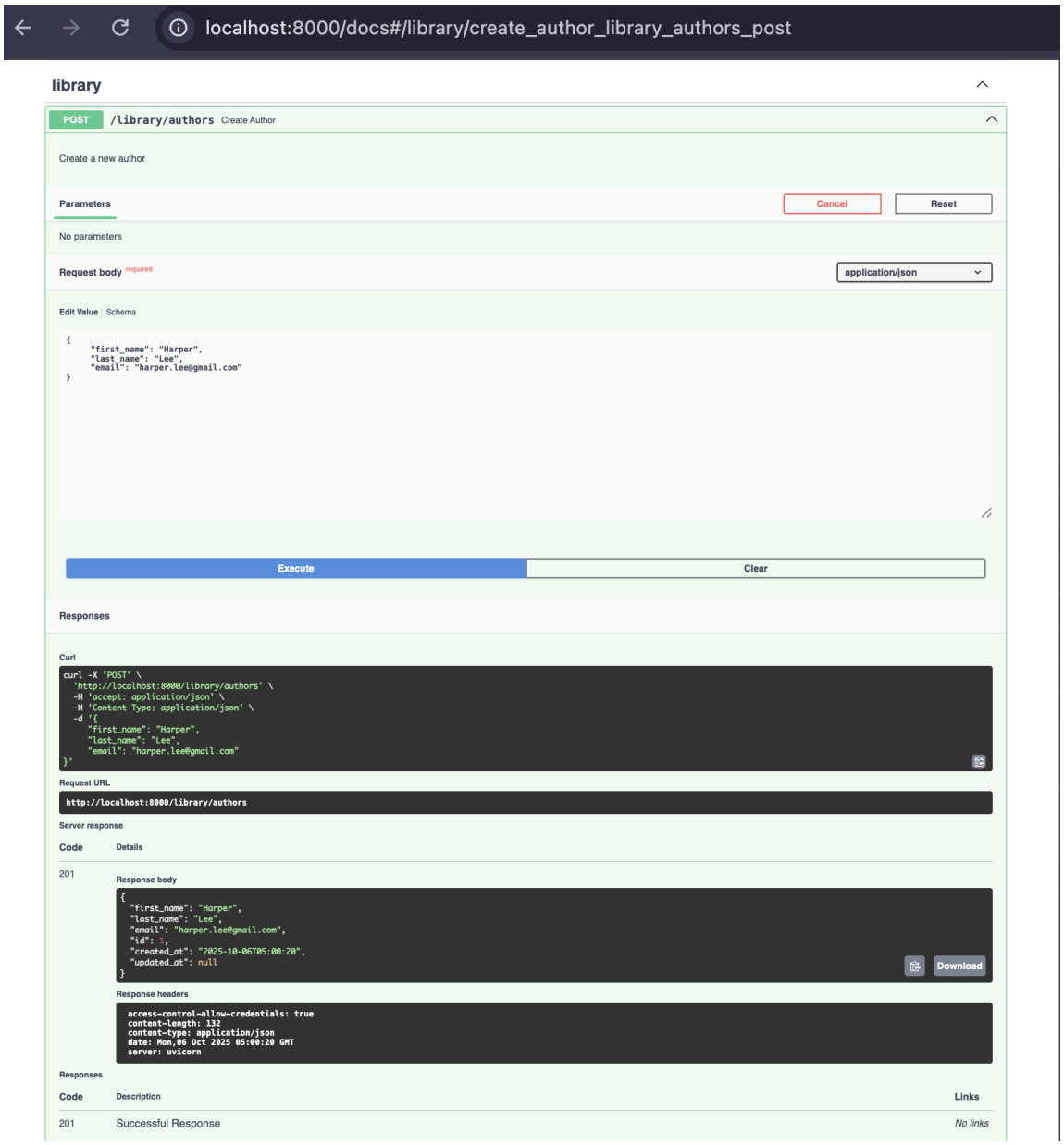



GET
/library/authors
Get Authors
Get all authors with pagination
Parameters
Cancel
Name
Description
page
integer
(query)
Page number
1
minimum: 1
per_page
integer
(query)
Items per page
10
maximum: 100
minimum: 1
Execute
Clear
Responses
Curl
curl -X 'GET' \
'http://localhost:8000/library/authors?page=1&per_page=10' \
-H 'accept: application/json'
Request URL
http://localhost:8000/library/authors?page=1&per_page=10
Server response
Code
Details
200
Response body
"updated_at": null
},
{
"first_name": "George",
"last_name": "Orwell",
"email": "george.orwell@gmail.com",
"id": 2,
"created_at": "2025-10-06T05:02:35",
"updated_at": null
},
{
"first_name": "J.K.",
"last_name": "Rowling",
"email": "jk.rowling@gmail.com",
"id": 3,
"created_at": "2025-10-06T05:03:00",
"updated_at": null
}
],
"total": 3,
"page": 1,
"per_page": 10,
"total_pages": 1
}
Download

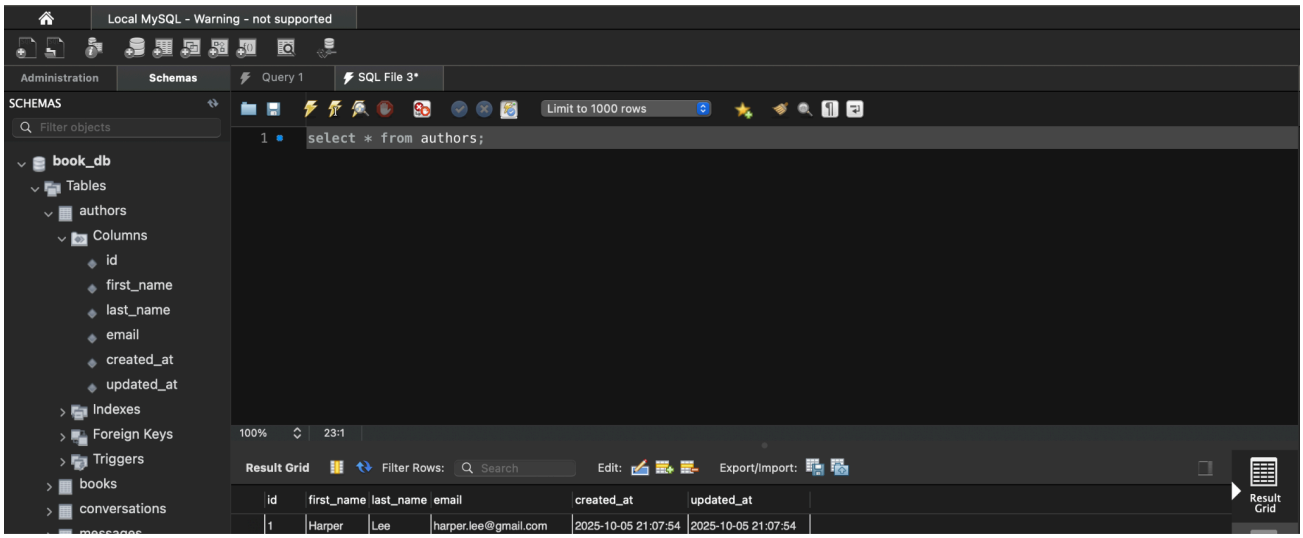
SQL Screenshot:



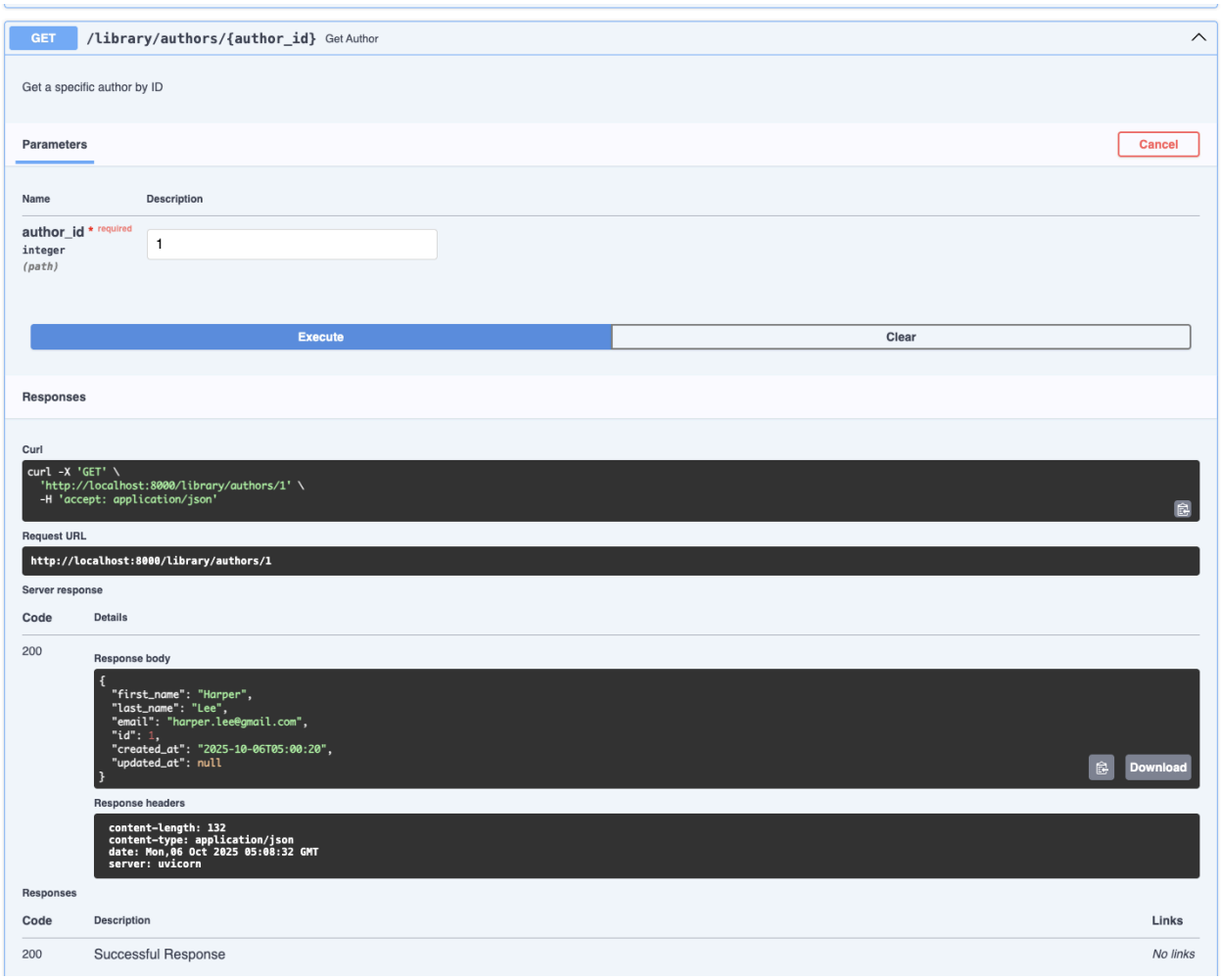
2. POST /library/authors - Create new author



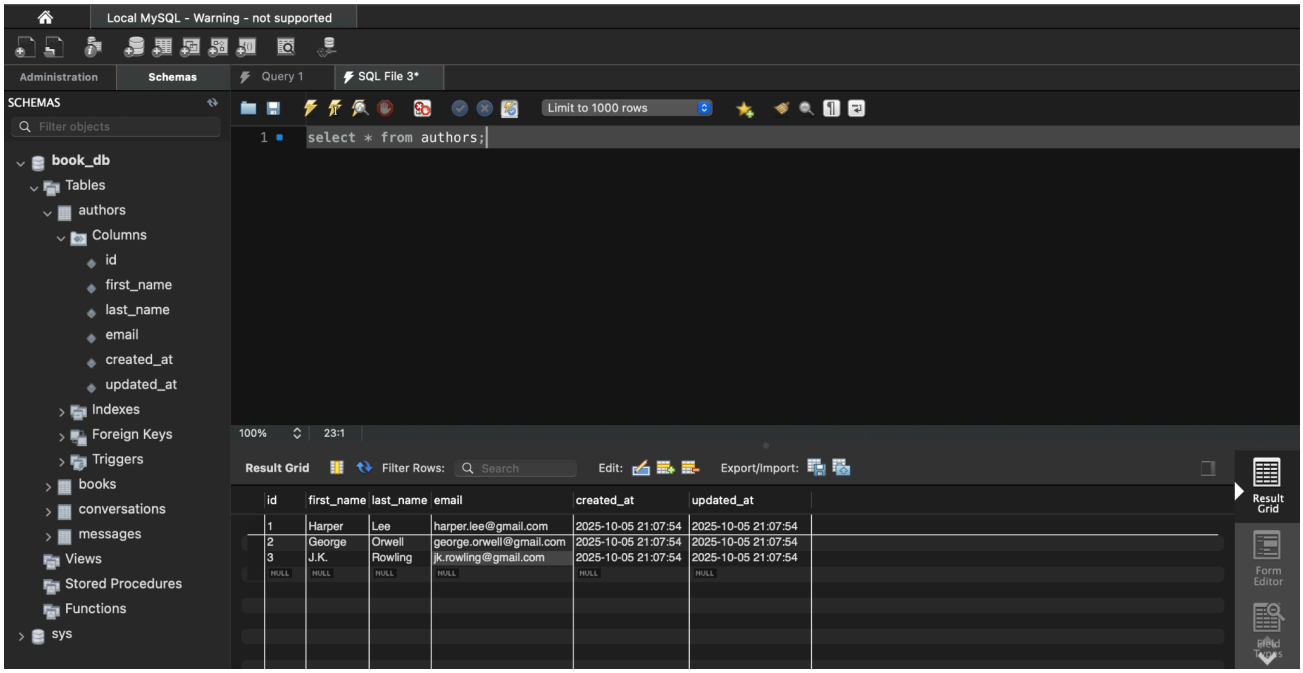
SQL Screenshots:



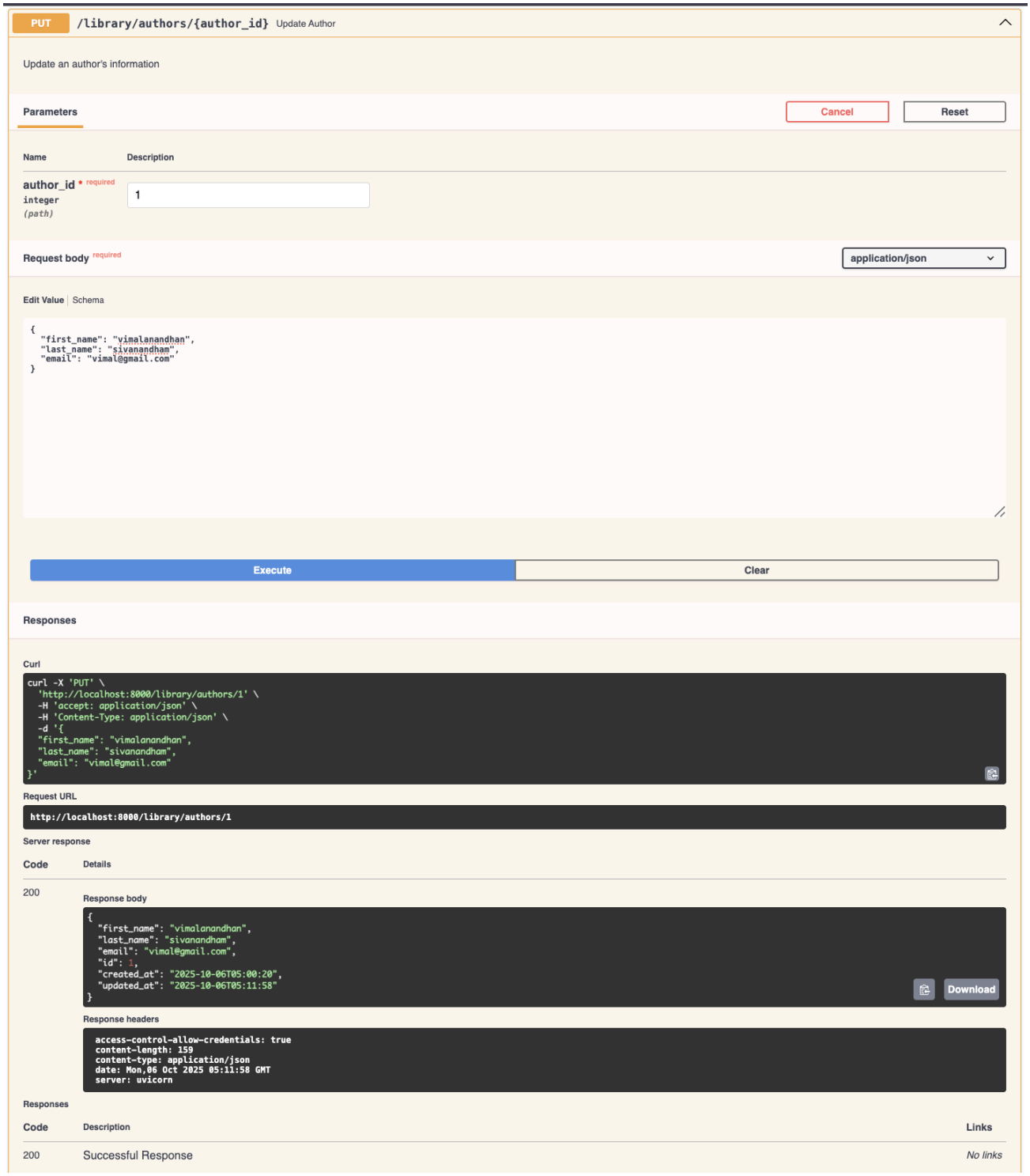
3. GET /library/authors/{author_id} - Get specific author



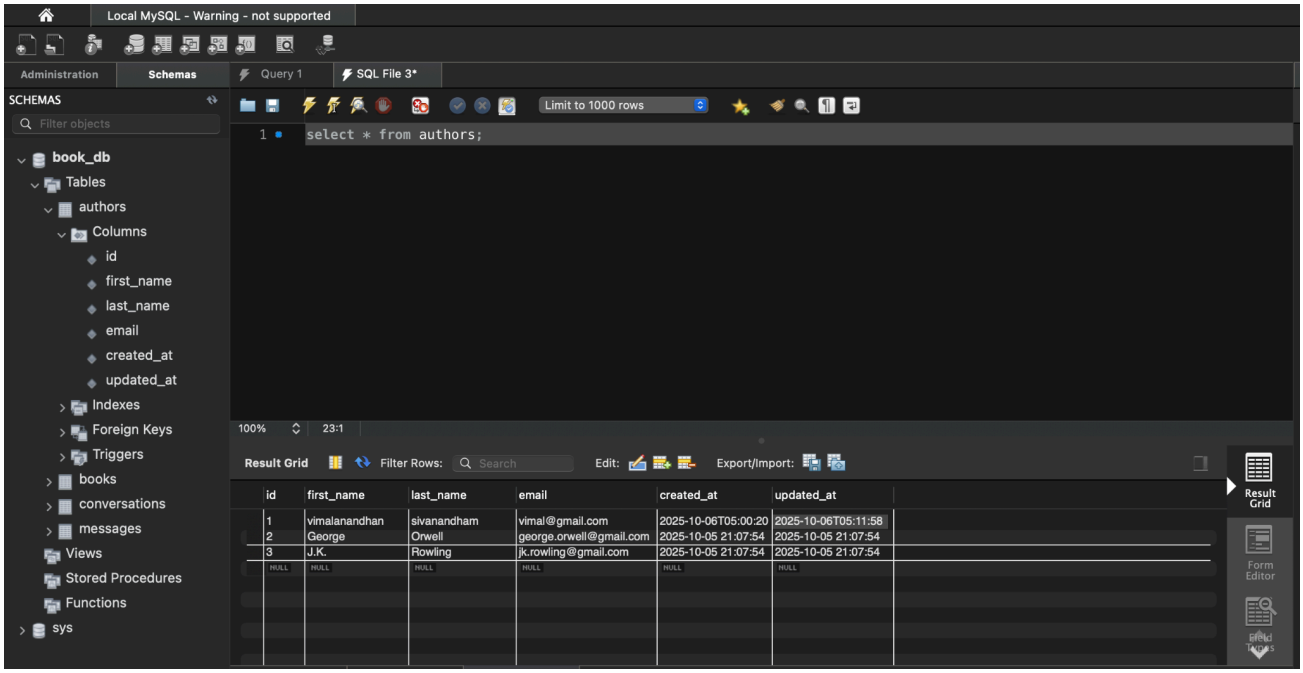
SQL:


Local MySQL - Warning - not supported
Administration
Schemas
Query 1
SQL File 3*
SCHEMAS
Filter objects
book_db
Tables
authors
Columns
id
first_name
last_name
email
created_at
updated_at
Indexes
Foreign Keys
Triggers
books
conversations
messages
Views
Stored Procedures
Functions
sys
Limit to 1000 rows
1
select * from authors;
100%
23:1
Result Grid
Filter Rows:
Search
Edit:
Export/Import:
id
first_name
last_name
email
created_at
updated_at
1
Harper
Lee
harper.lee@gmail.com
2025-10-05 21:07:54
2025-10-05 21:07:54
2
George
Orwell
george.orwell@gmail.com
2025-10-05 21:07:54
2025-10-05 21:07:54
3
J.K.
Rowling
jk.rowling@gmail.com
2025-10-05 21:07:54
2025-10-05 21:07:54
NULL
Result Grid
Form Editor
Field Types

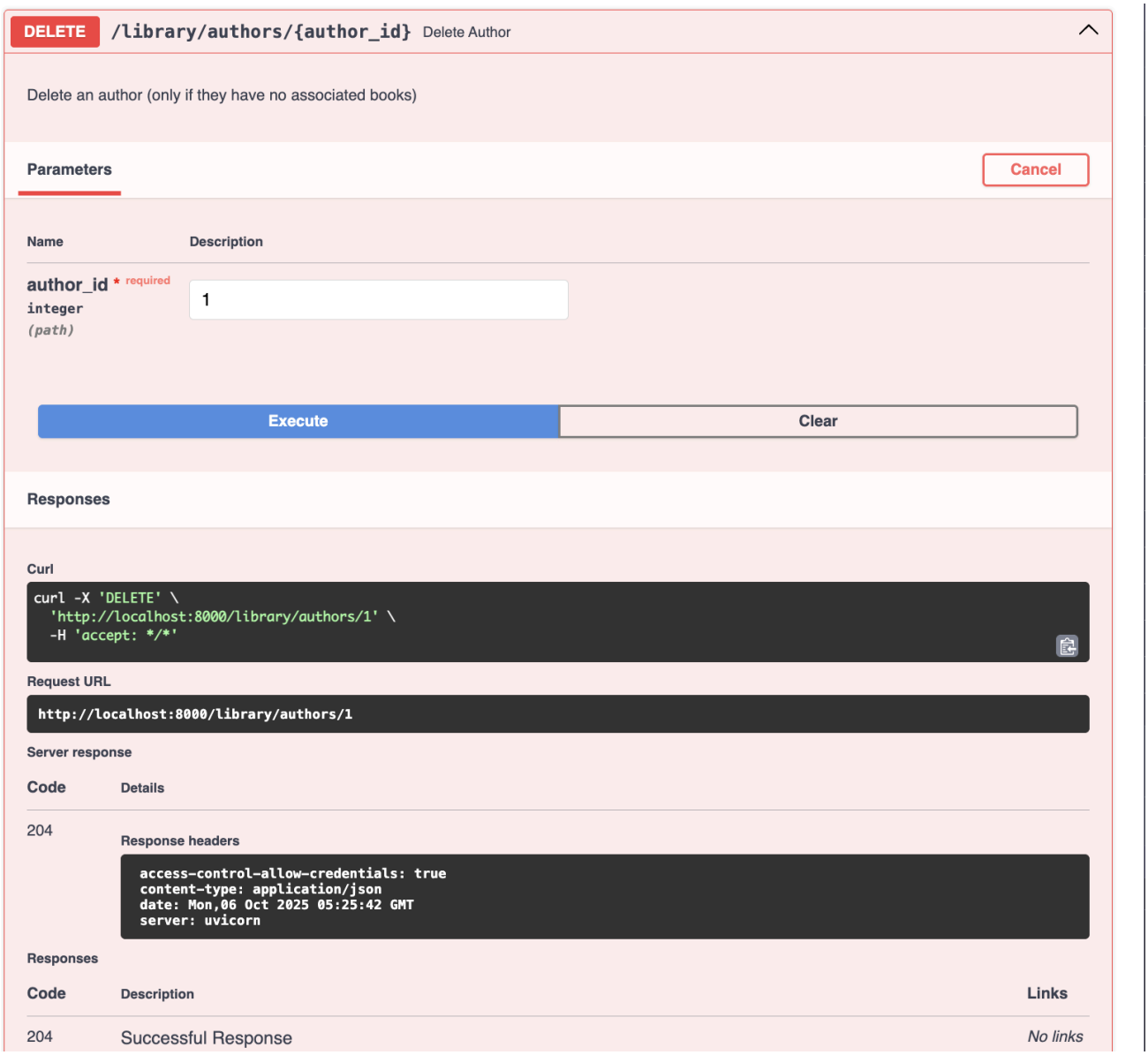
4. PUT /library/authors/{author_id} - Update author:



SQL:

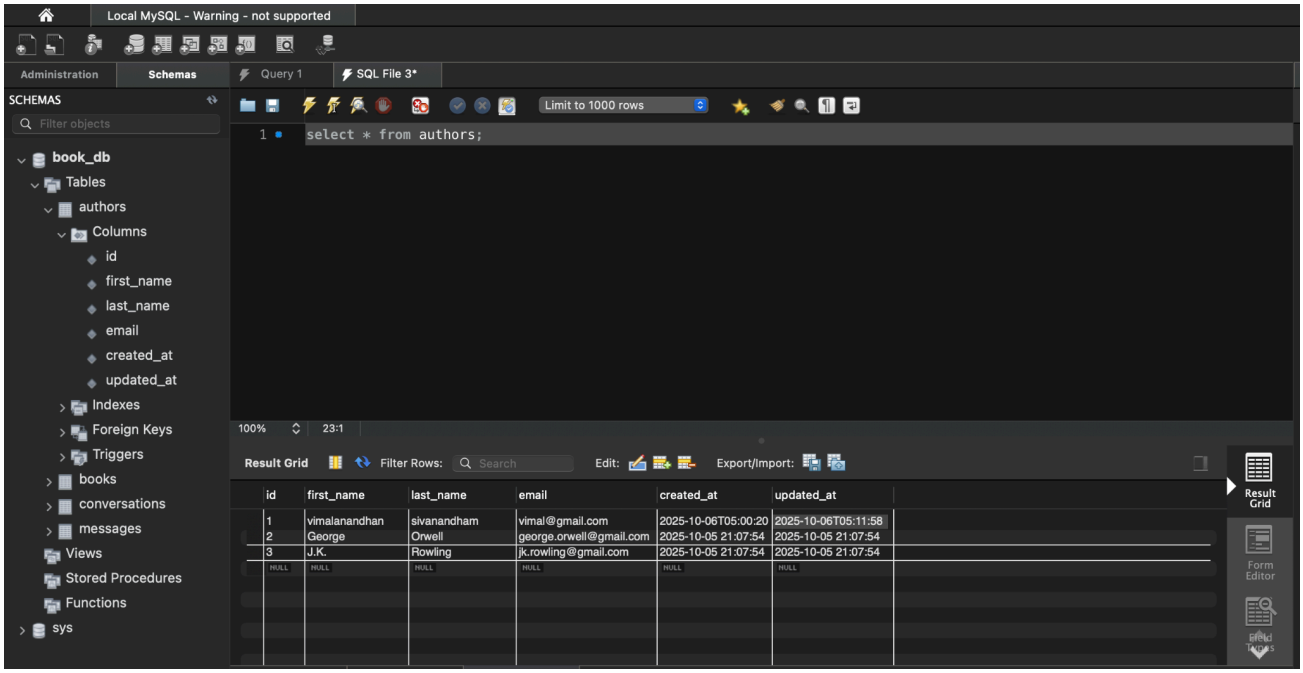

Local MySQL - Warning - not supported
Administration
Schemas
Query 1
SQL File 3*
SCHEMAS
Filter objects
book_db
Tables
authors
Columns
id
first_name
last_name
email
created_at
updated_at
Indexes
Foreign Keys
Triggers
books
conversations
messages
Views
Stored Procedures
Functions
sys
Limit to 1000 rows
1 select * from authors;
100%
23:1
Result Grid
Filter Rows:
Search
Edit:
Export/Import:
id
first_name
last_name
email
created_at
updated_at
1
vimalanandhan
sivanandham
vimal@gmail.com
2025-10-06T05:00:20
2025-10-06T05:11:58
2
George
Orwell
george.orwell@gmail.com
2025-10-05 21:07:54
2025-10-05 21:07:54
3
J.K.
Rowling
jk.rowling@gmail.com
2025-10-05 21:07:54
2025-10-05 21:07:54
NULL
Result Grid
Form Editor

5. DELETE /library/authors/{author_id} Delete Author

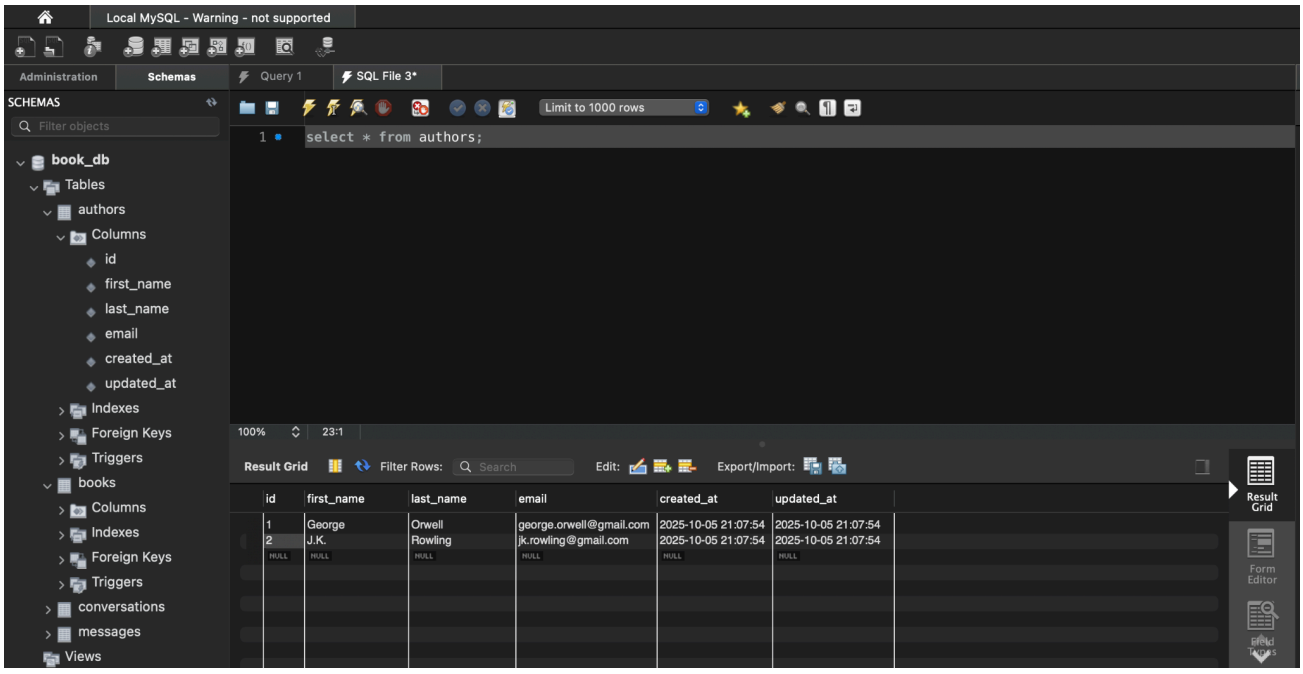


SQL

BEFORE DELETE:

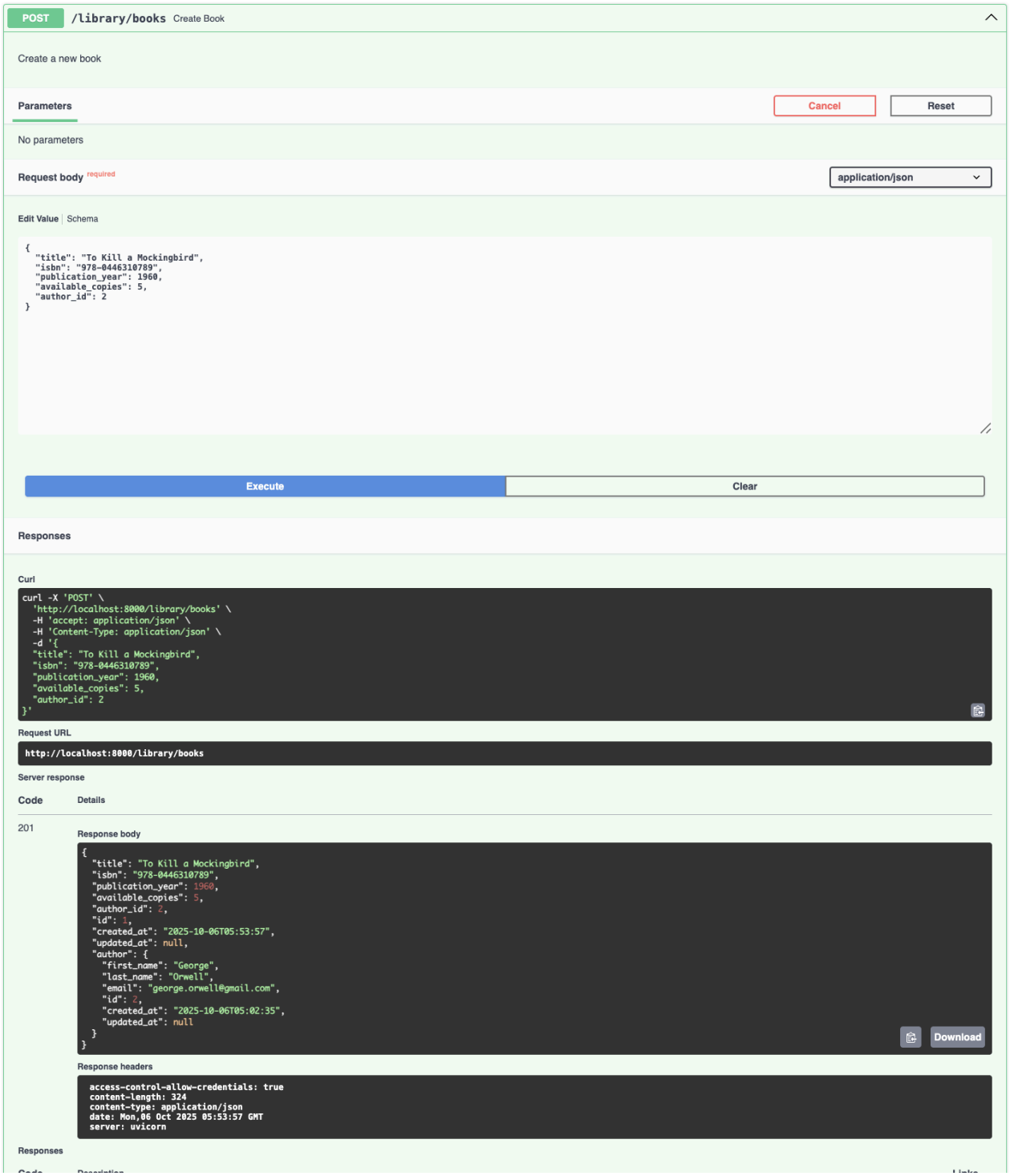


AFTER DELETE:



Books API Endpoint

6.POST /library/books - Create new book



Create a new book

Parameters Cancel Reset

No parameters

Request body required application/json

Edit Value | Schema

```
{
  "title": "To Kill a Mockingbird",
  "isbn": "978-0446310789",
  "publication_year": 1960,
  "available_copies": 5,
  "author_id": 2
}
```

Execute Clear

Responses

Curl

```
curl -X 'POST' \
  'http://localhost:8000/library/books' \
  -H 'accept: application/json' \
  -H 'Content-Type: application/json' \
  -d '{
  "title": "To Kill a Mockingbird",
  "isbn": "978-0446310789",
  "publication_year": 1960,
  "available_copies": 5,
  "author_id": 2
}'
```

Request URL

```
http://localhost:8000/library/books
```

Server response

Code	Details
201	**Response body**

```
{
  "title": "To Kill a Mockingbird",
  "isbn": "978-0446310789",
  "publication_year": 1960,
  "available_copies": 5,
  "author_id": 2,
  "id": 1,
  "created_at": "2025-10-06T05:53:57",
  "updated_at": null,
  "author": {
    "first_name": "George",
    "last_name": "Orwell",
    "email": "george.orwell@gmail.com",
    "id": 2,
    "created_at": "2025-10-06T05:02:35",
    "updated_at": null
  }
}
```

Download

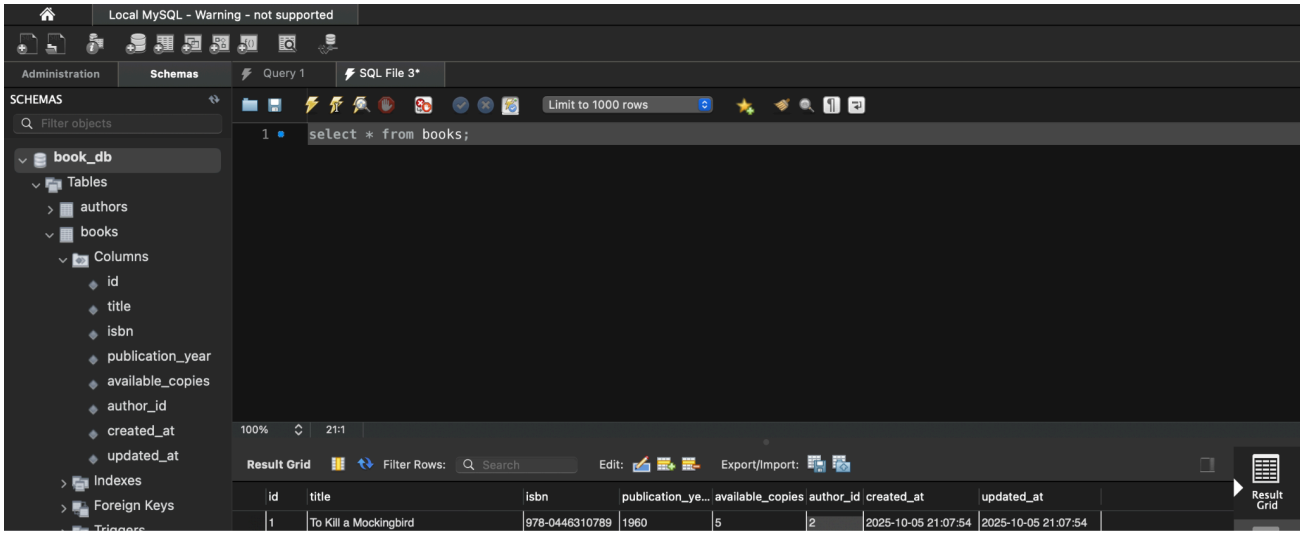
Response headers

```
access-control-allow-credentials: true
content-length: 324
content-type: application/json
date: Mon,06 Oct 2025 05:53:57 GMT
server: uvicorn
```

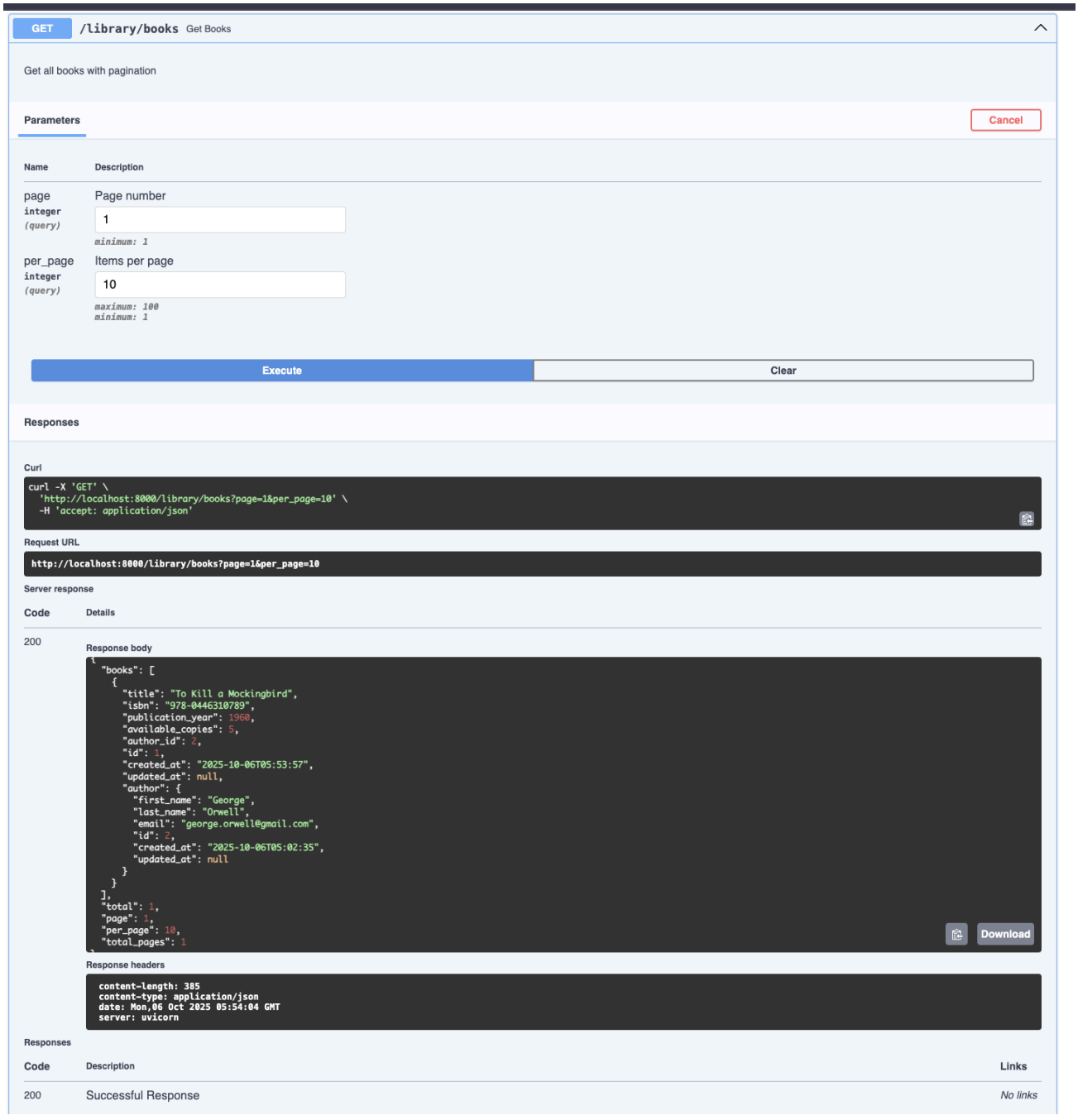
Responses

Code	Description	Links
201	Successful Response	*No links*

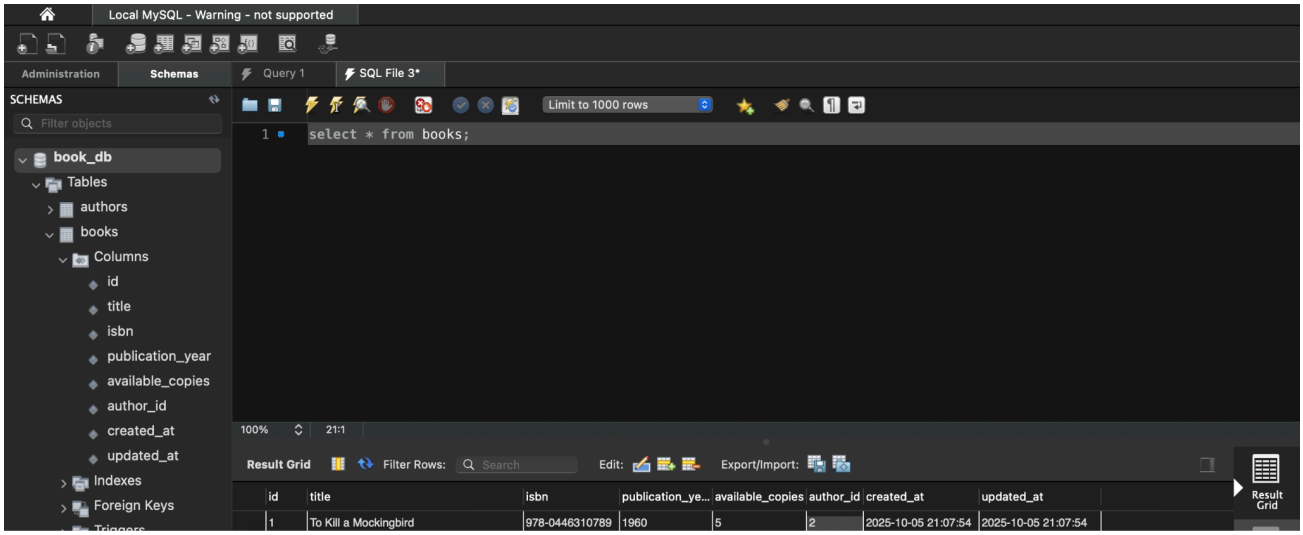
SQL:


Local MySQL - Warning - not supported
Administration
Schemas
Query 1
SQL File 3*
SCHEMAS
Filter objects
book_db
Tables
authors
books
Columns
id
title
isbn
publication_year
available_copies
author_id
created_at
updated_at
Indexes
Foreign Keys
Limit to 1000 rows
1 select * from books;
100%
21:1
Result Grid
Filter Rows:
Search
Edit:
Export/Import:
id
title
isbn
publication_ye...
available_copies
author_id
created_at
updated_at
1
To Kill a Mockingbird
978-0446310789
1960
5
2
2025-10-05 21:07:54
2025-10-05 21:07:54
Result Grid

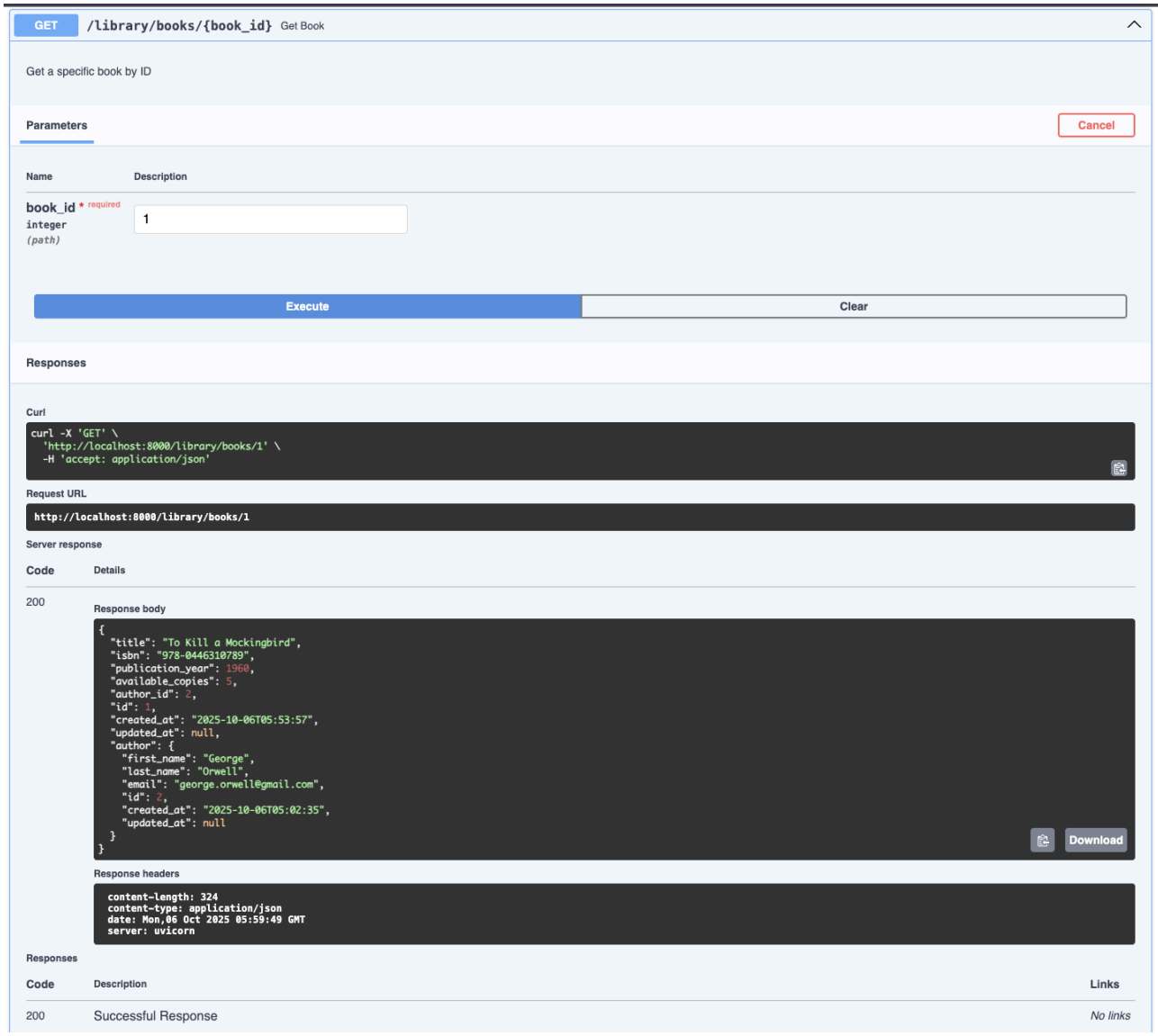
7.GET /library/books - Get all books



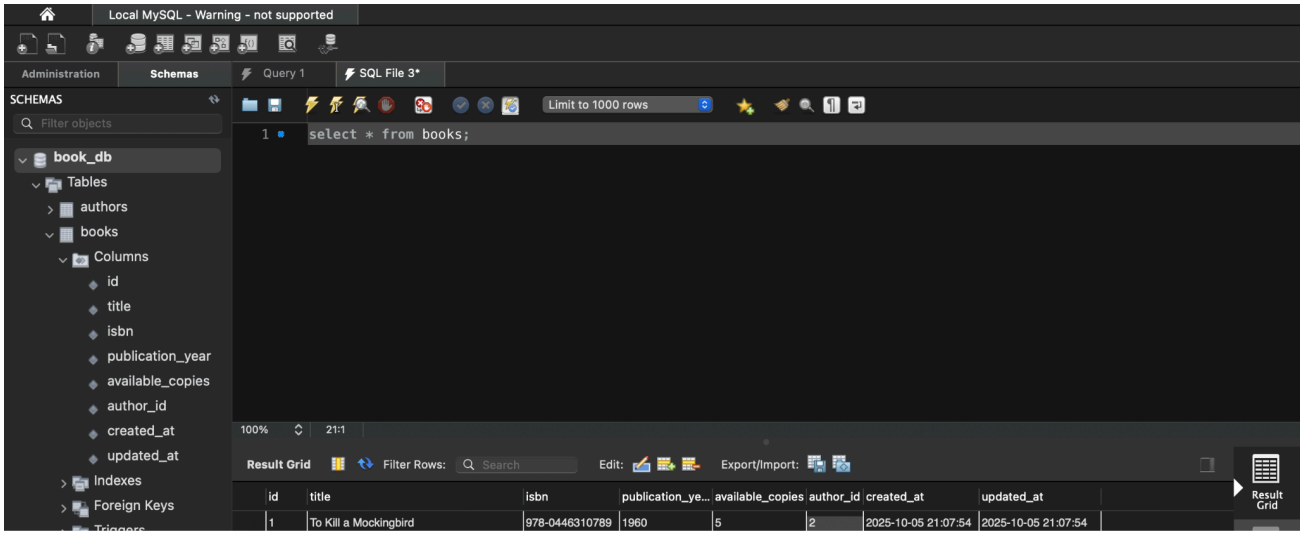
SQL:


Local MySQL - Warning - not supported
Administration
Schemas
Query 1
SQL File 3*
SCHEMAS
Filter objects
book_db
Tables
authors
books
Columns
id
title
isbn
publication_year
available_copies
author_id
created_at
updated_at
Indexes
Foreign Keys
Limit to 1000 rows
1 select * from books;
100%
21:1
Result Grid
Filter Rows:
Search
Edit:
Export/Import:
id
title
isbn
publication_ye...
available_copies
author_id
created_at
updated_at
1
To Kill a Mockingbird
978-0446310789
1960
5
2
2025-10-05 21:07:54
2025-10-05 21:07:54
Result Grid

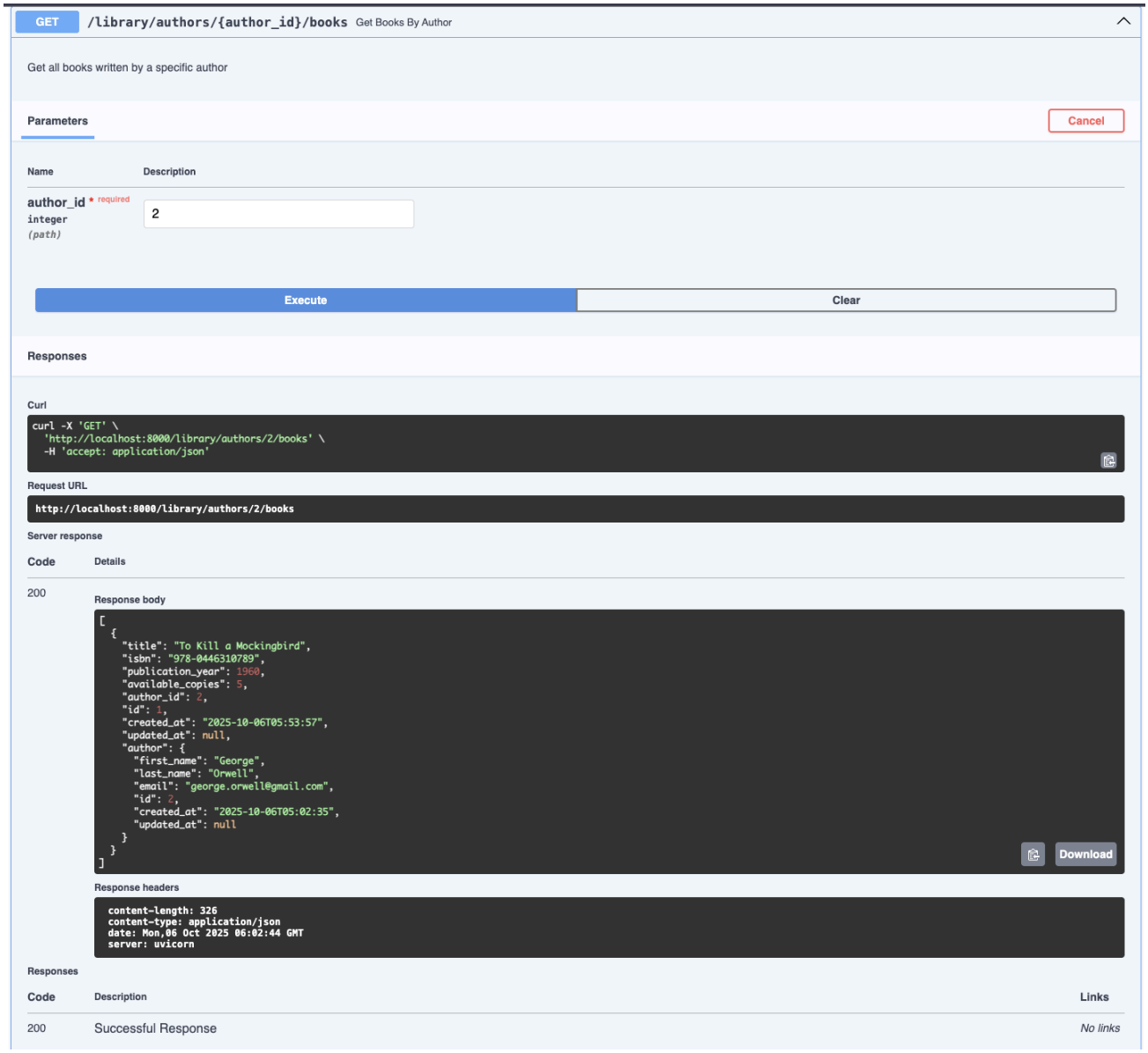
8.GET /library/books/{book_id} - Get specific book



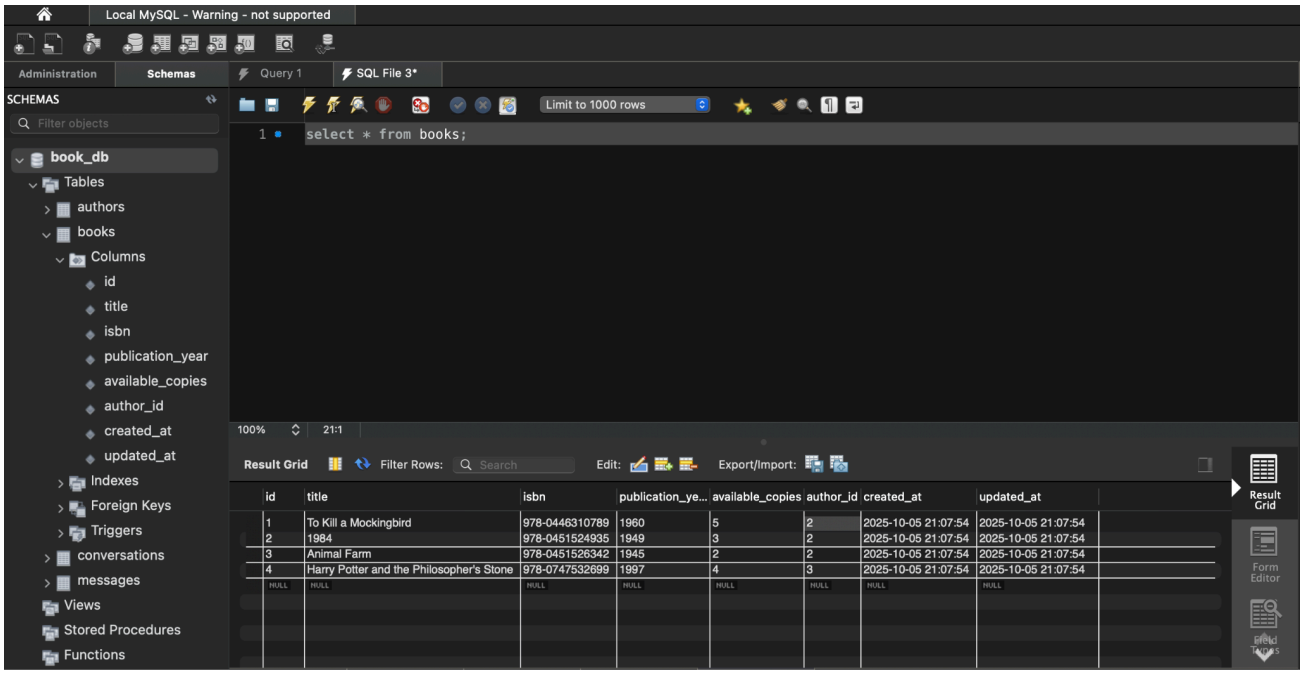
SQL:


Local MySQL - Warning - not supported
Administration
Schemas
Query 1
SQL File 3*
SCHEMAS
Filter objects
book_db
Tables
authors
books
Columns
id
title
isbn
publication_year
available_copies
author_id
created_at
updated_at
Indexes
Foreign Keys
Limit to 1000 rows
1 select * from books;
100%
21:1
Result Grid
Filter Rows:
Search
Edit:
Export/Import:
id
title
isbn
publication_ye...
available_copies
author_id
created_at
updated_at
1
To Kill a Mockingbird
978-0446310789
1960
5
2
2025-10-05 21:07:54
2025-10-05 21:07:54
Result Grid

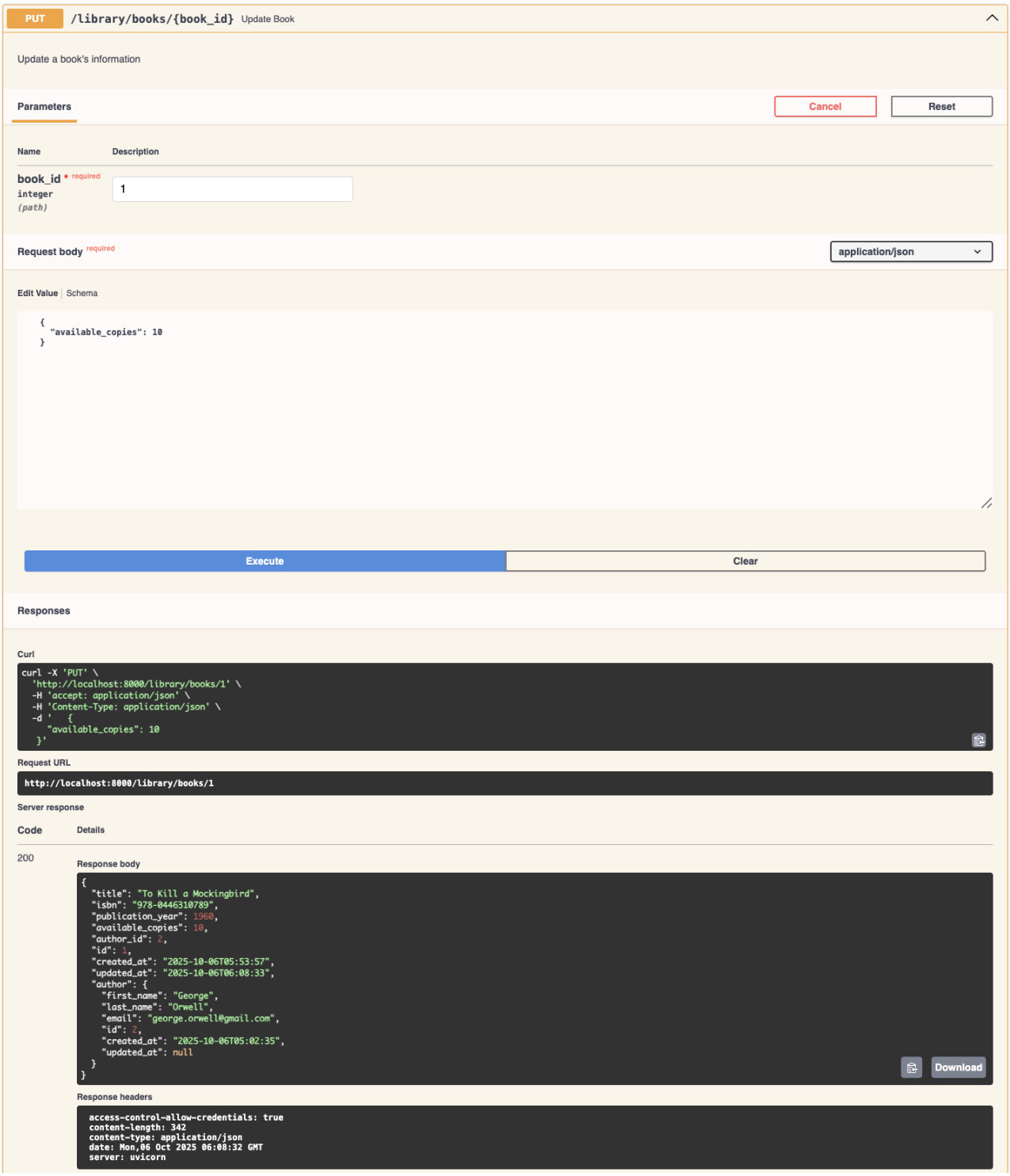
9.GET /library/authors/{author_id}/books - Get books by author



SQL:


Local MySQL - Warning - not supported
Administration
Schemas
Query 1
SQL File 3*
SCHEMAS
Filter objects
book_db
Tables
authors
books
Columns
id
title
isbn
publication_year
available_copies
author_id
created_at
updated_at
Indexes
Foreign Keys
Triggers
conversations
messages
Views
Stored Procedures
Functions
Limit to 1000 rows
1 select * from books;
100%
21:1
Result Grid
Filter Rows:
Search
Edit:
Export/Import:
id title isbn publication_ye... available_copies author_id created_at updated_at
1 To Kill a Mockingbird 978-0446310789 1960 5 2 2025-10-05 21:07:54 2025-10-05 21:07:54
2 1984 978-0451524935 1949 3 2 2025-10-05 21:07:54 2025-10-05 21:07:54
3 Animal Farm 978-0451526342 1945 2 2 2025-10-05 21:07:54 2025-10-05 21:07:54
4 Harry Potter and the Philosopher's Stone 978-0747532699 1997 4 3 2025-10-05 21:07:54 2025-10-05 21:07:54
Result Grid
Form Editor

10.PUT /library/books/{book_id} - Update book



Name	Description
book_id * required integer *(path)*	1

Request body required — application/json

Edit Value | Schema

```
{
  "available_copies": 10
}
```

Execute | Clear

Responses

Curl

```
curl -X 'PUT' \
  'http://localhost:8000/library/books/1' \
  -H 'accept: application/json' \
  -H 'Content-Type: application/json' \
  -d '  {
    "available_copies": 10
  }'
```

Request URL

```
http://localhost:8000/library/books/1
```

Server response

Code	Details
200	Response body

```
{
  "title": "To Kill a Mockingbird",
  "isbn": "978-0446310789",
  "publication_year": 1960,
  "available_copies": 10,
  "author_id": 2,
  "id": 1,
  "created_at": "2025-10-06T05:53:57",
  "updated_at": "2025-10-06T06:08:33",
  "author": {
    "first_name": "George",
    "last_name": "Orwell",
    "email": "george.orwell@gmail.com",
    "id": 2,
    "created_at": "2025-10-06T05:02:35",
    "updated_at": null
  }
}
```

Download

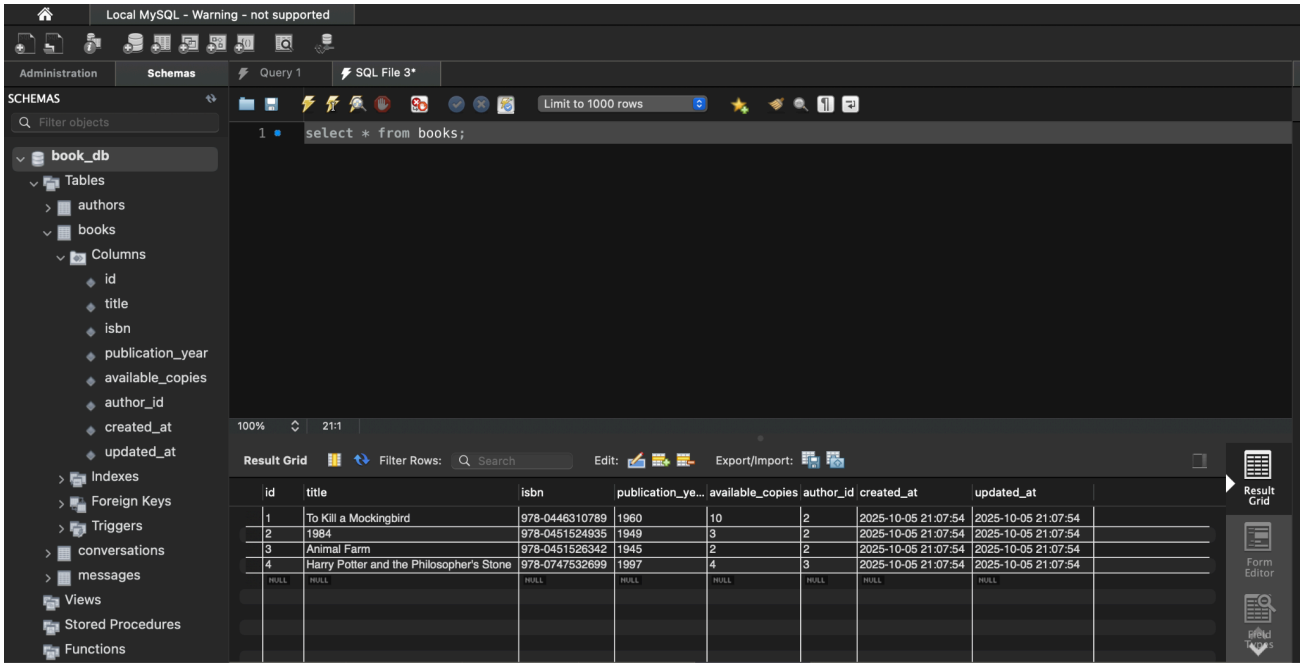
Response headers

```
access-control-allow-credentials: true
content-length: 342
content-type: application/json
date: Mon,06 Oct 2025 06:08:32 GMT
server: uvicorn
```

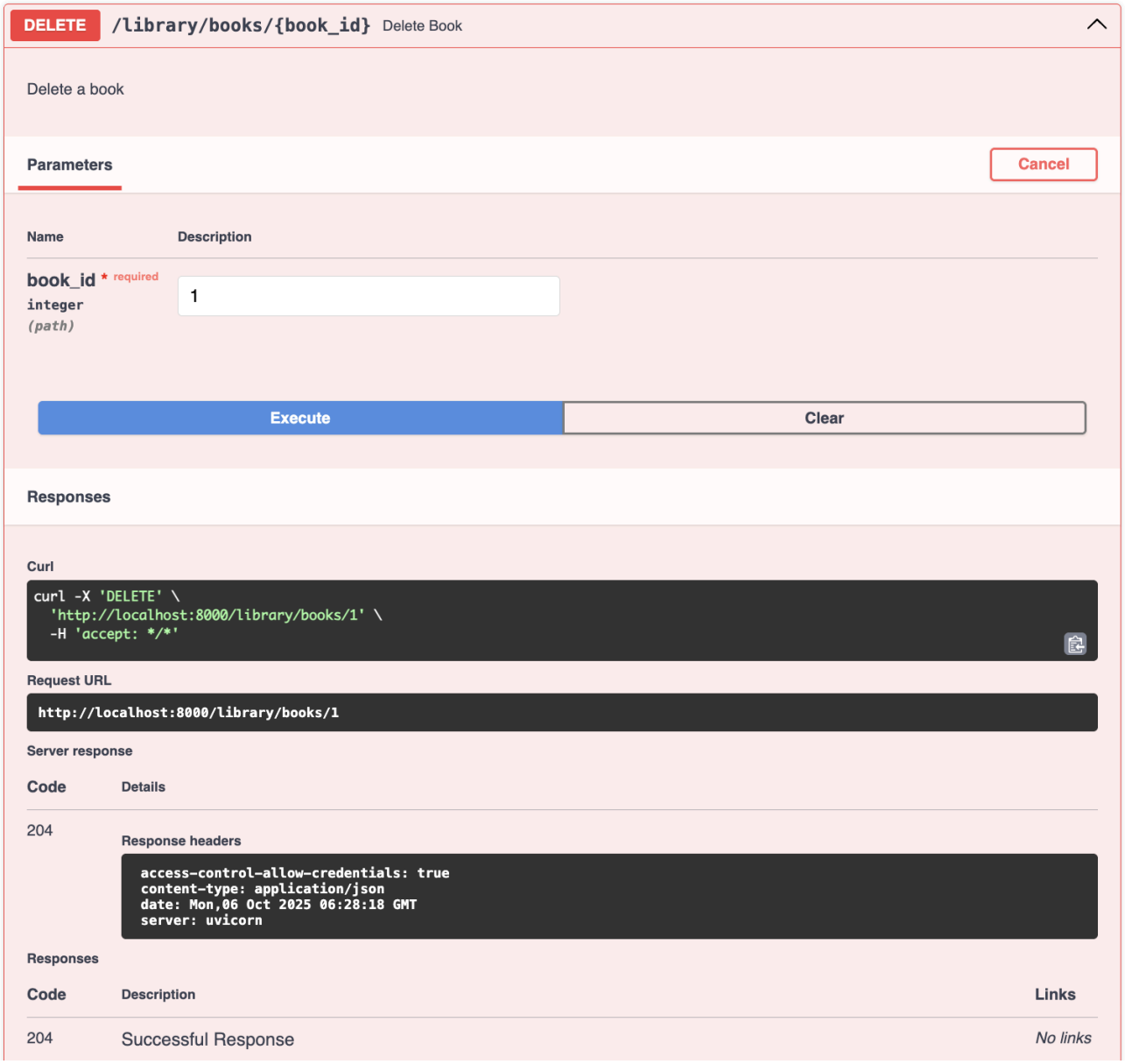
Responses

Code	Description	Links
200	Successful Response	*No links*

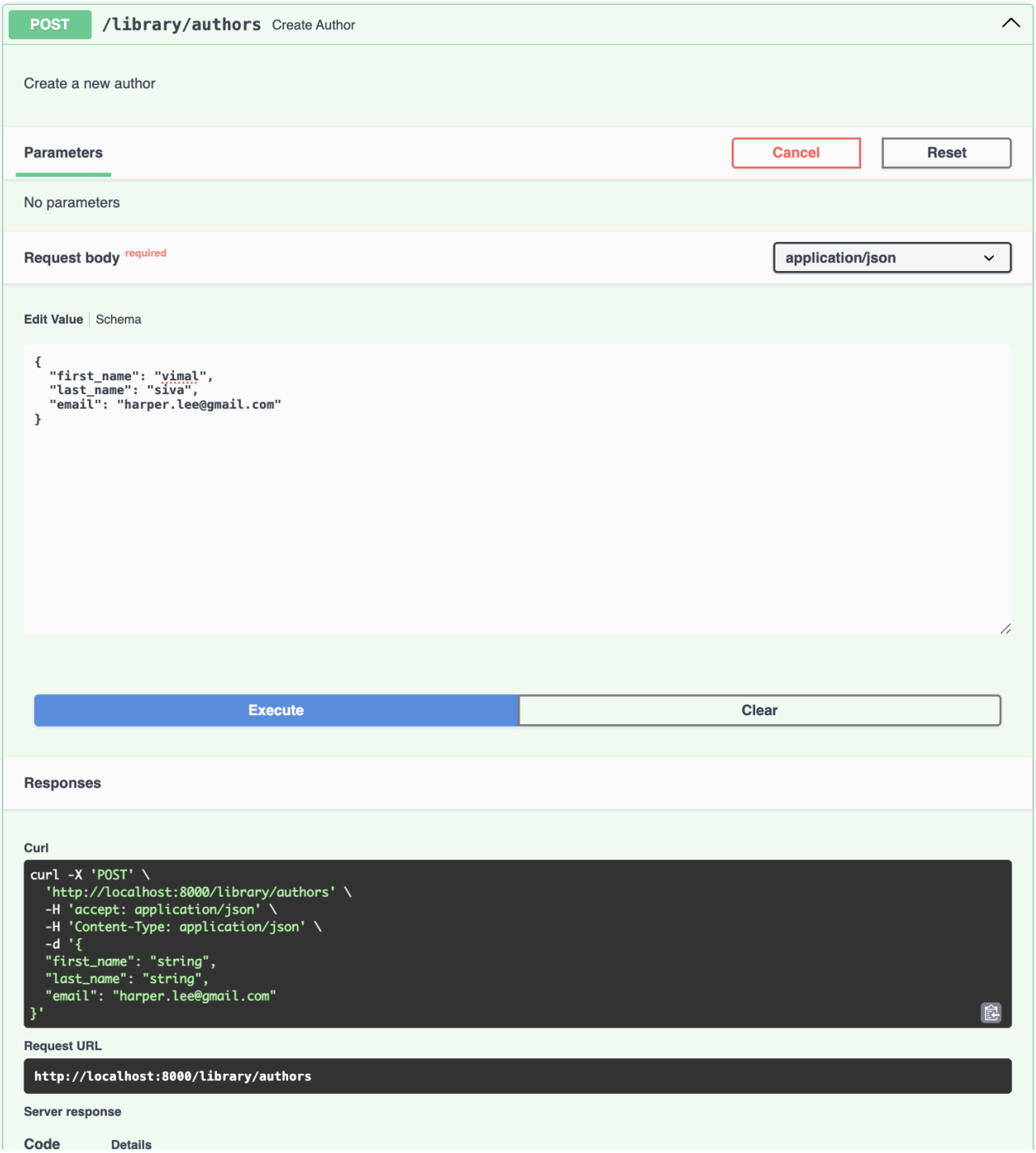
SQL:

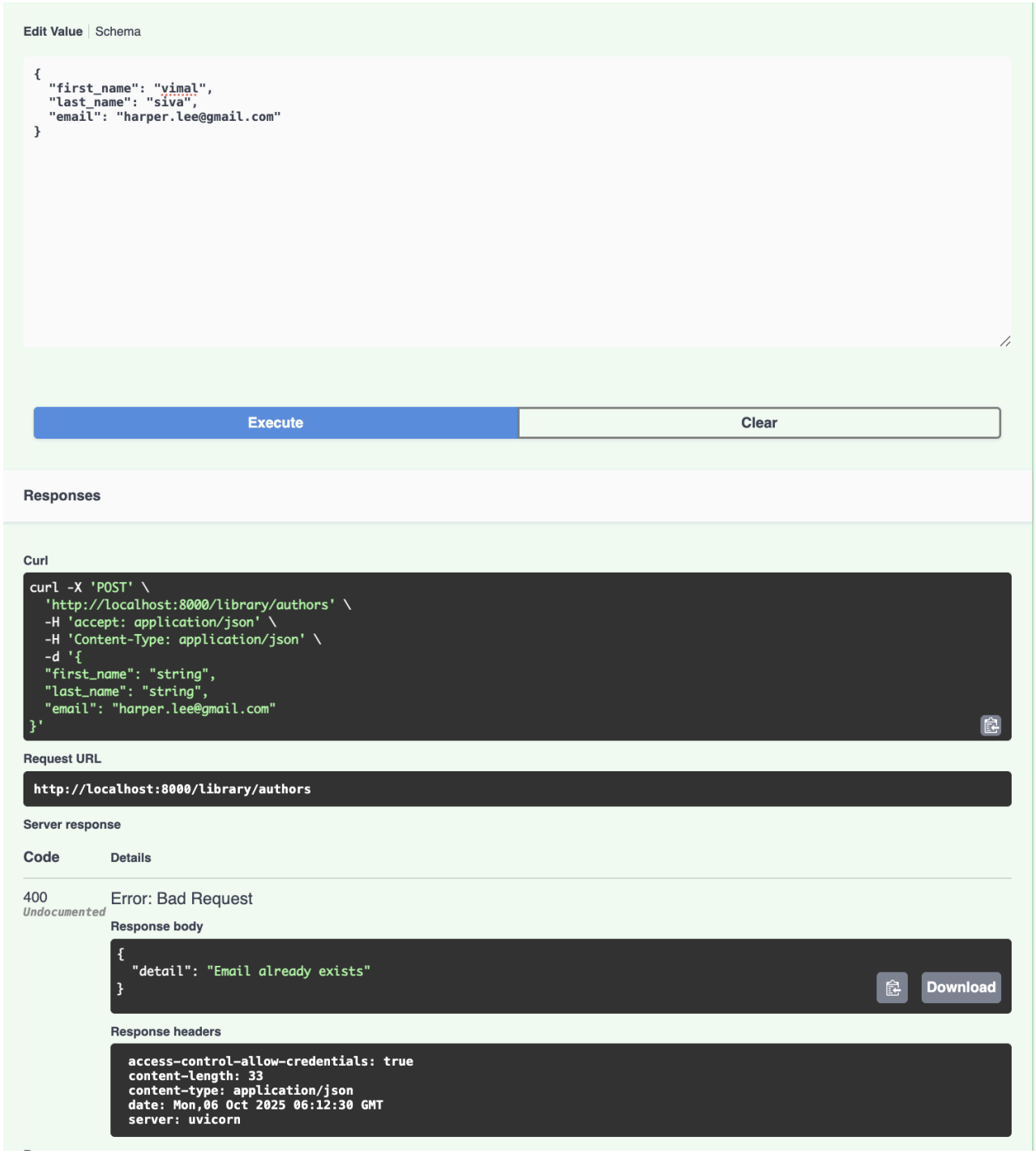

Local MySQL - Warning - not supported
Administration
Schemas
Query 1
SQL File 3*
SCHEMAS
Filter objects
book_db
Tables
authors
books
Columns
id
title
isbn
publication_year
available_copies
author_id
created_at
updated_at
Indexes
Foreign Keys
Triggers
conversations
messages
Views
Stored Procedures
Functions
Limit to 1000 rows
1 select * from books;
100%
21:1
Result Grid
Filter Rows:
Search
Edit:
Export/Import:
id title isbn publication_ye... available_copies author_id created_at updated_at
1 To Kill a Mockingbird 978-0446310789 1960 10 2 2025-10-05 21:07:54 2025-10-05 21:07:54
2 1984 978-0451524935 1949 3 2 2025-10-05 21:07:54 2025-10-05 21:07:54
3 Animal Farm 978-0451526342 1945 2 2 2025-10-05 21:07:54 2025-10-05 21:07:54
4 Harry Potter and the Philosopher's Stone 978-0747532699 1997 4 3 2025-10-05 21:07:54 2025-10-05 21:07:54
Result Grid
Form Editor

11.DELETE /library/books/{book_id} - Delete book



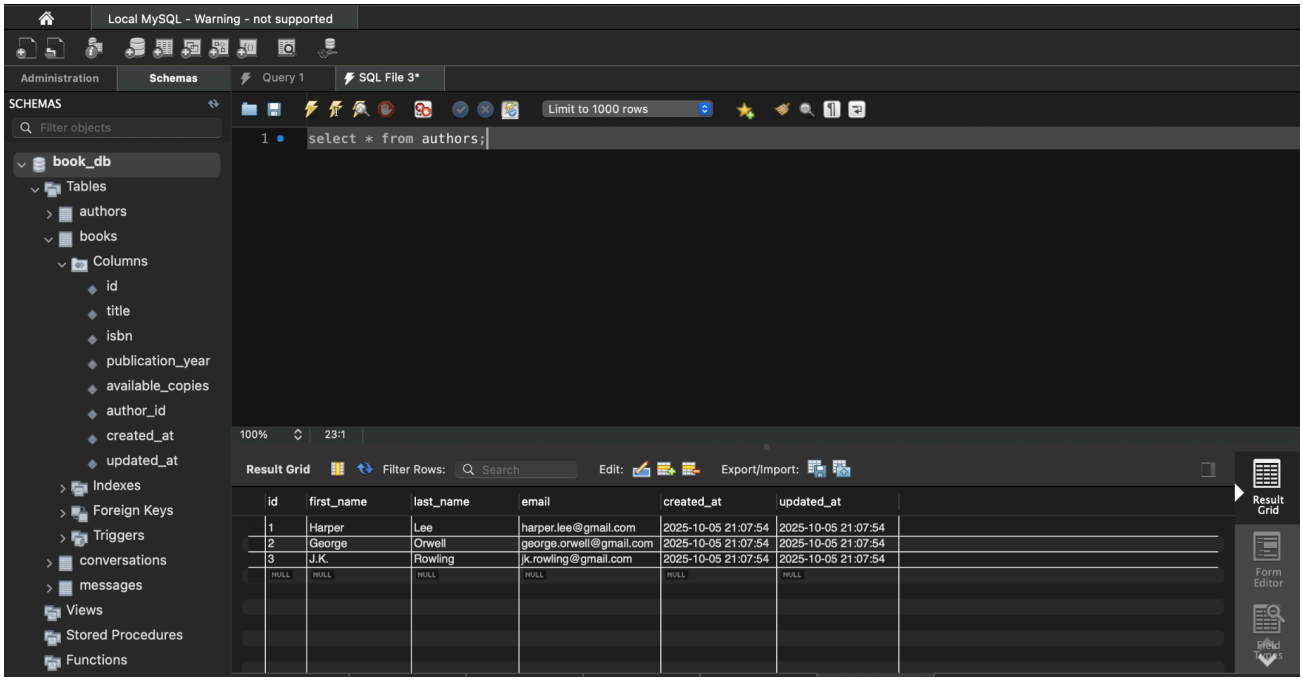
12.Test Duplicate Email



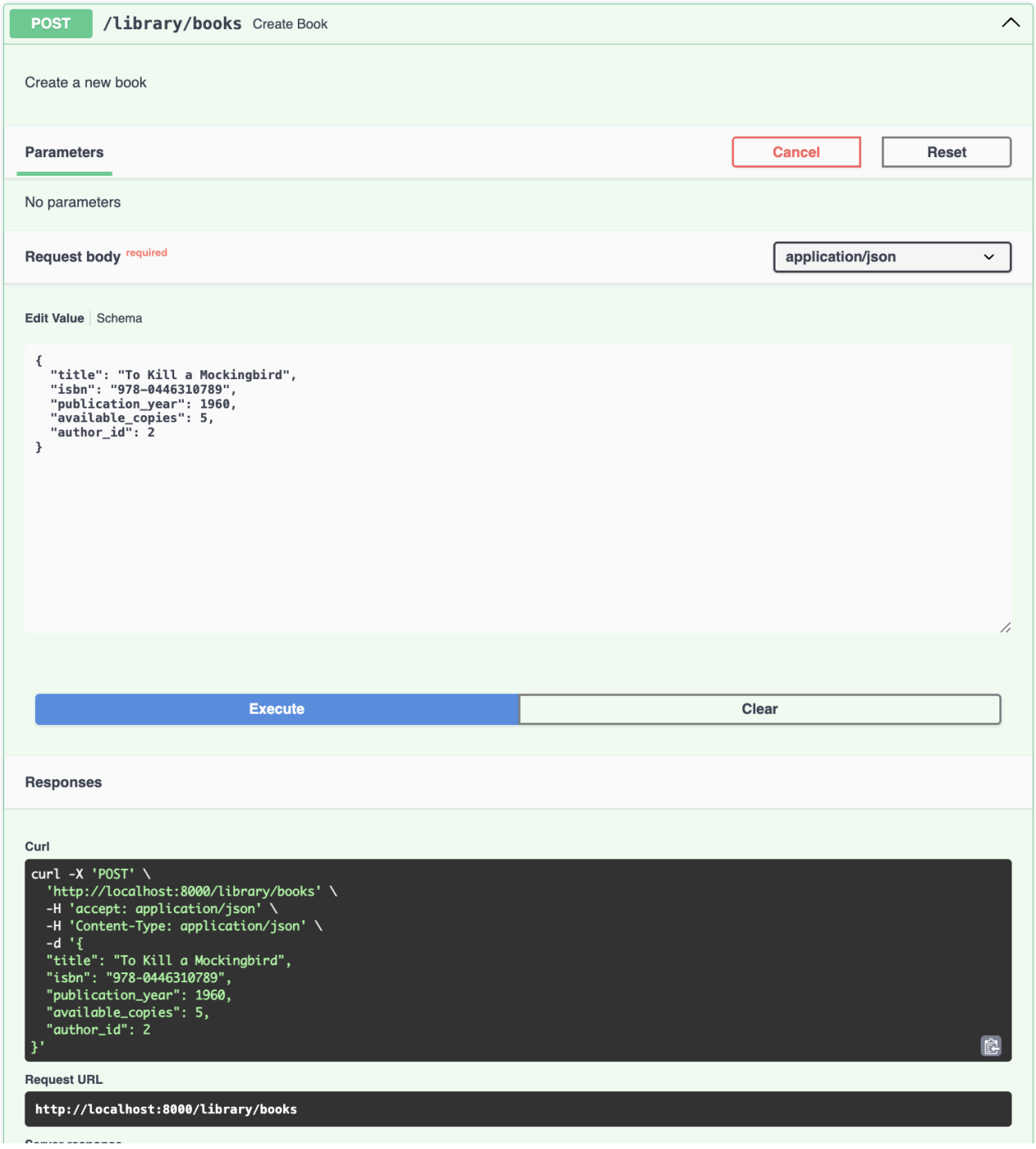


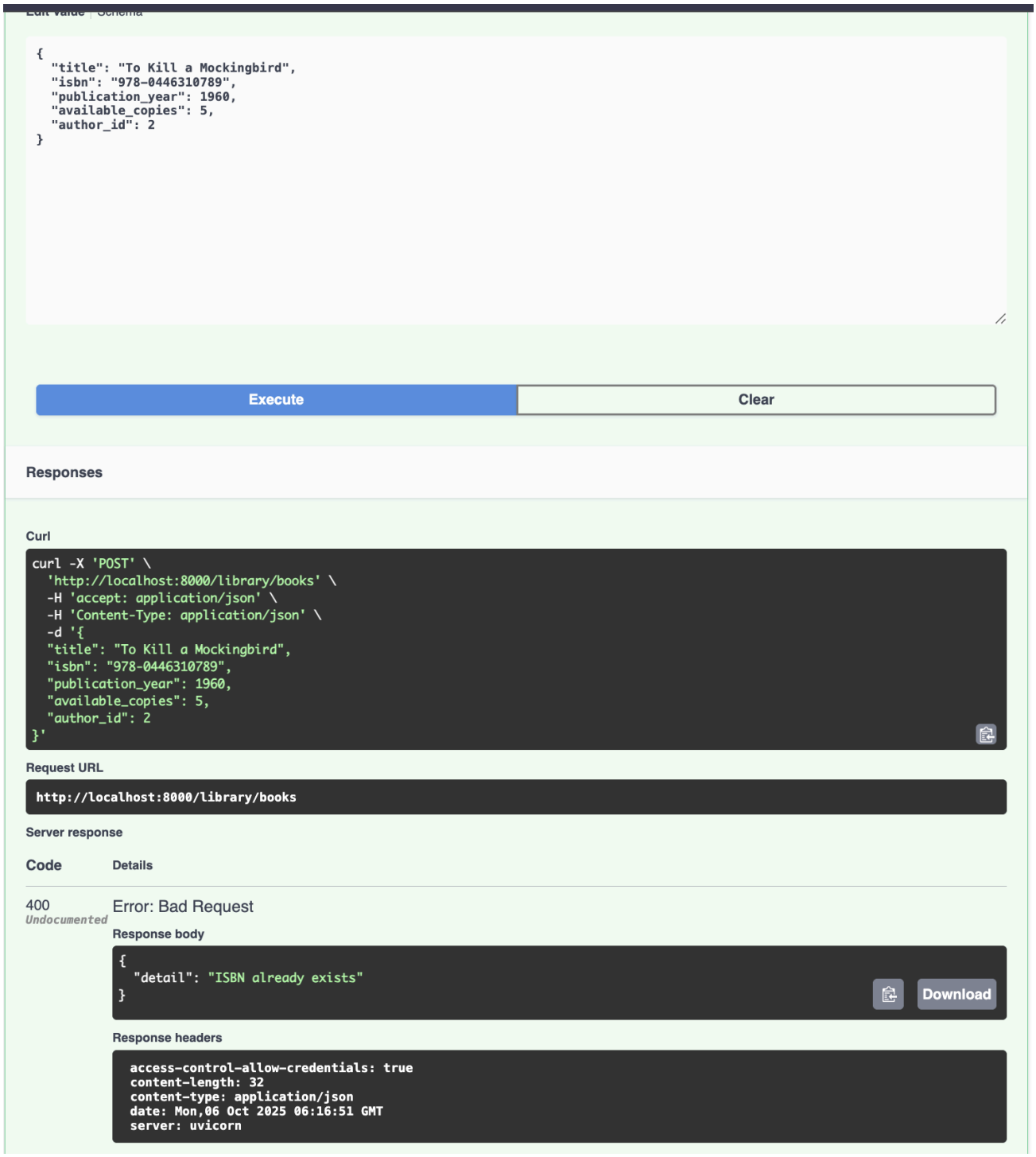
We used the same email address before which is already available in the db as below. That's the reason the status code and response body is throwing errors.

SQL:


Local MySQL - Warning - not supported
Administration
Schemas
Query 1
SQL File 3*
SCHEMAS
Filter objects
book_db
Tables
authors
books
Columns
id
title
isbn
publication_year
available_copies
author_id
created_at
updated_at
Indexes
Foreign Keys
Triggers
conversations
messages
Views
Stored Procedures
Functions
Limit to 1000 rows
1 select * from authors;
100%
23:1
Result Grid
Filter Rows:
Search
Edit:
Export/Import:
id
first_name
last_name
email
created_at
updated_at
1 Harper Lee harper.lee@gmail.com 2025-10-05 21:07:54 2025-10-05 21:07:54
2 George Orwell george.orwell@gmail.com 2025-10-05 21:07:54 2025-10-05 21:07:54
3 J.K. Rowling jk.rowling@gmail.com 2025-10-05 21:07:54 2025-10-05 21:07:54
Result Grid
Form Editor

13. Test Duplicate ISBN

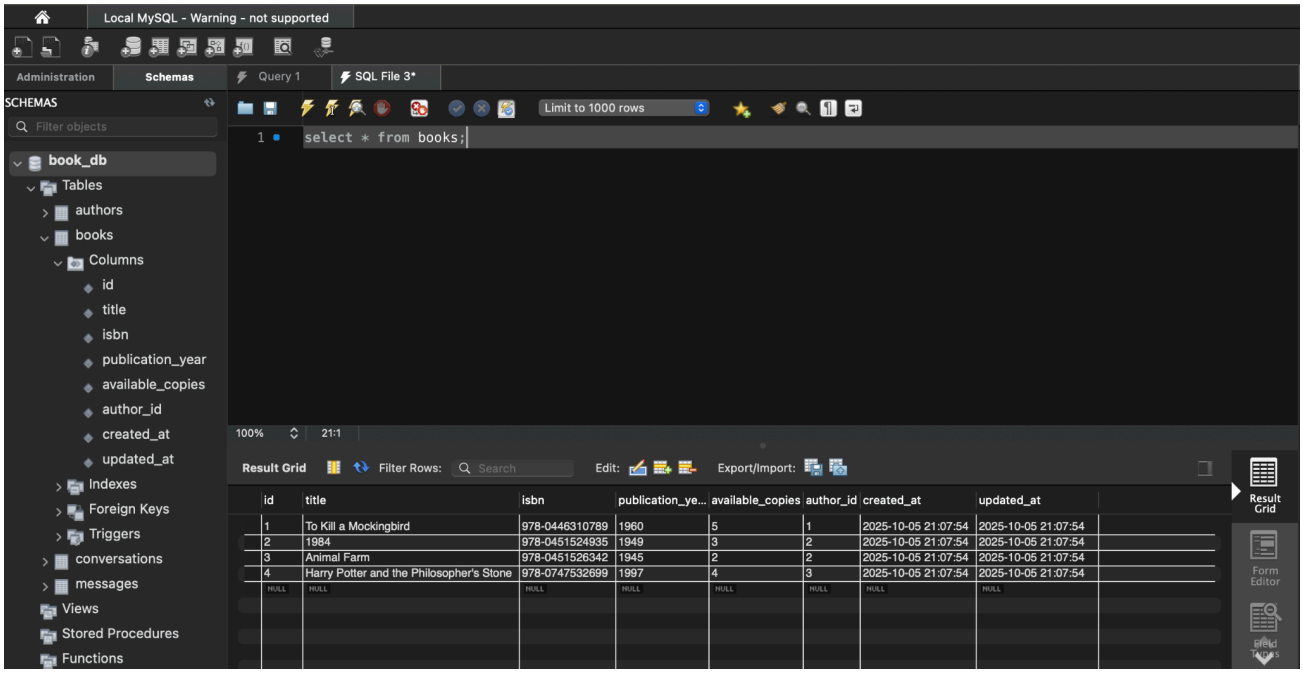




We used the same ISBN before which is already available in the db as below.

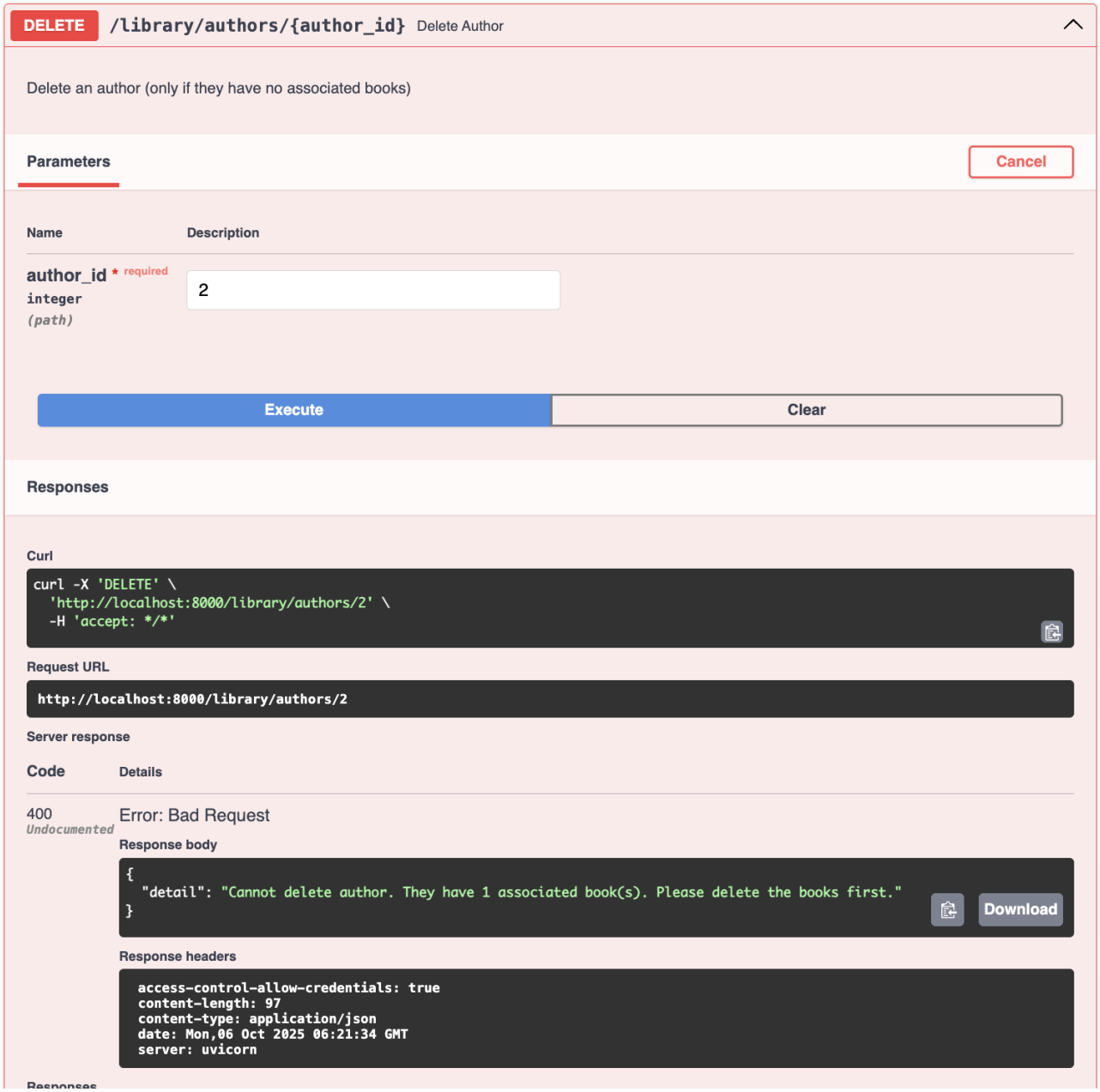
That's the reason the status code and response body is throwing errors.

SQL:


Local MySQL - Warning - not supported
Administration
Schemas
Query 1
SQL File 3*
SCHEMAS
Filter objects
book_db
Tables
authors
books
Columns
id
title
isbn
publication_year
available_copies
author_id
created_at
updated_at
Indexes
Foreign Keys
Triggers
conversations
messages
Views
Stored Procedures
Functions
Limit to 1000 rows
1 select * from books;
100%
21:1
Result Grid
Filter Rows:
Search
Edit:
Export/Import:
id title isbn publication_ye... available_copies author_id created_at updated_at
1 To Kill a Mockingbird 978-0446310789 1960 5 1 2025-10-05 21:07:54 2025-10-05 21:07:54
2 1984 978-0451524935 1949 3 2 2025-10-05 21:07:54 2025-10-05 21:07:54
3 Animal Farm 978-0451526342 1945 2 2 2025-10-05 21:07:54 2025-10-05 21:07:54
4 Harry Potter and the Philosopher's Stone 978-0747532699 1997 4 3 2025-10-05 21:07:54 2025-10-05 21:07:54
Result Grid
Form Editor
Field Types

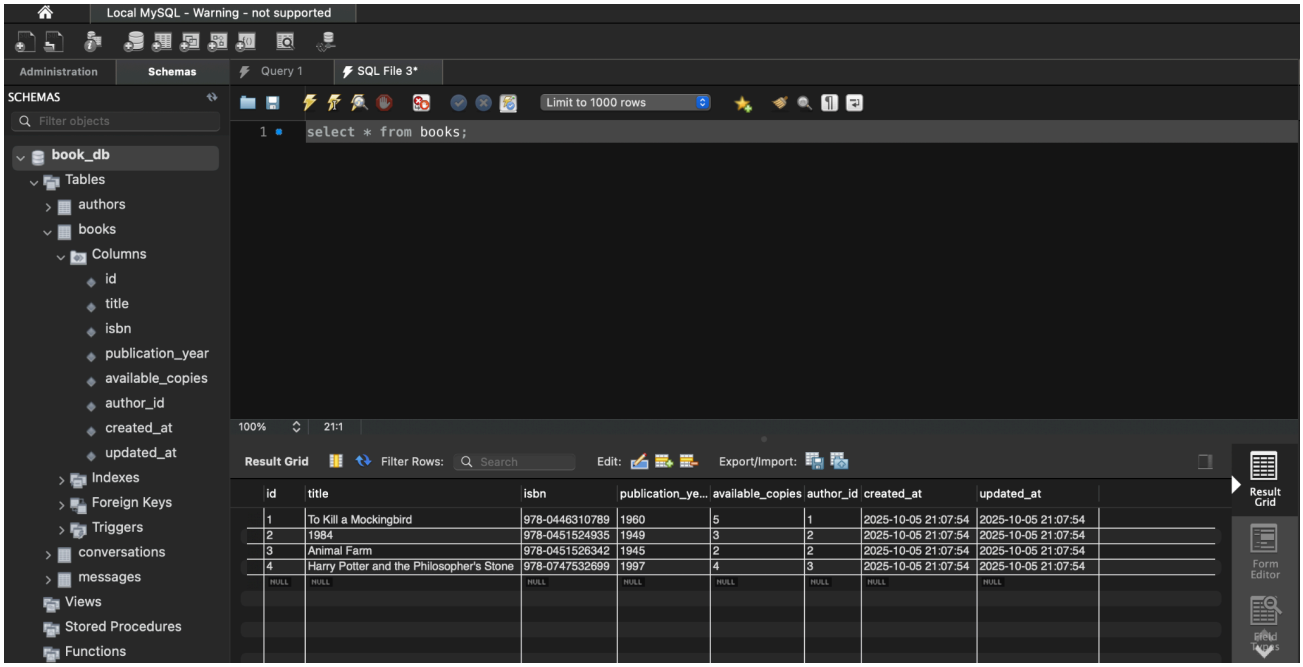
14.Test Author Deletion with Books - Try deleting an author who has books

---

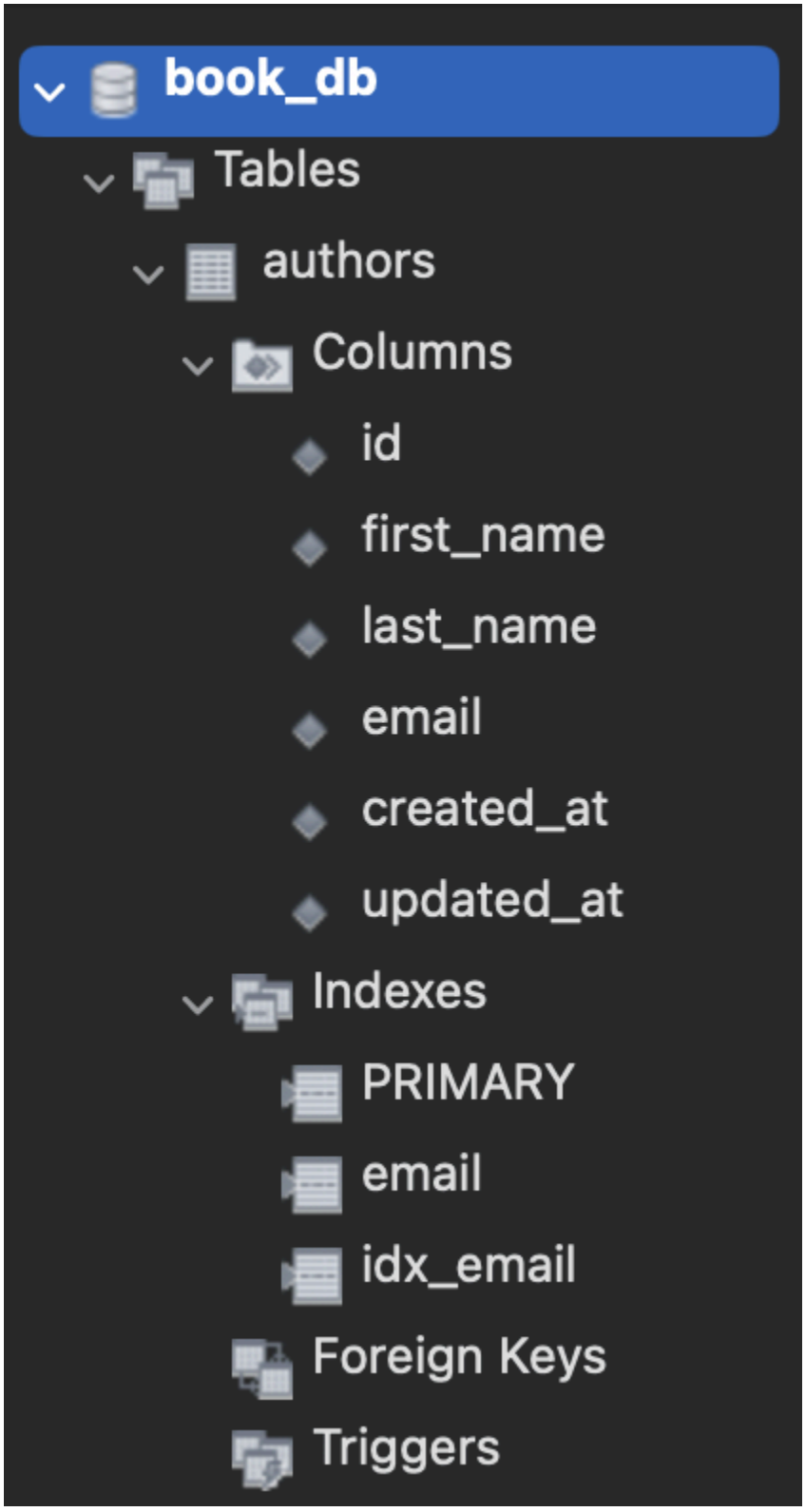


We tried to delete the author who has the books tagged under their id so ,we won't be able to delete it, we need to delete the book first as available in the db as below. That's the reason the status code and response body is throwing errors.

SQL:


Local MySQL - Warning - not supported
Administration
Schemas
Query 1
SQL File 3*
SCHEMAS
Filter objects
book_db
Tables
authors
books
Columns
id
title
isbn
publication_year
available_copies
author_id
created_at
updated_at
Indexes
Foreign Keys
Triggers
conversations
messages
Views
Stored Procedures
Functions
Limit to 1000 rows
1 select * from books;
100%
21:1
Result Grid
Filter Rows:
Search
Edit:
Export/Import:
id
title
isbn
publication_ye...
available_copies
author_id
created_at
updated_at
1 To Kill a Mockingbird 978-0446310789 1960 5 1 2025-10-05 21:07:54 2025-10-05 21:07:54
2 1984 978-0451524935 1949 3 2 2025-10-05 21:07:54 2025-10-05 21:07:54
3 Animal Farm 978-0451526342 1945 2 2 2025-10-05 21:07:54 2025-10-05 21:07:54
4 Harry Potter and the Philosopher's Stone 978-0747532699 1997 4 3 2025-10-05 21:07:54 2025-10-05 21:07:54
Result Grid
Form Editor

SQL TABLES:

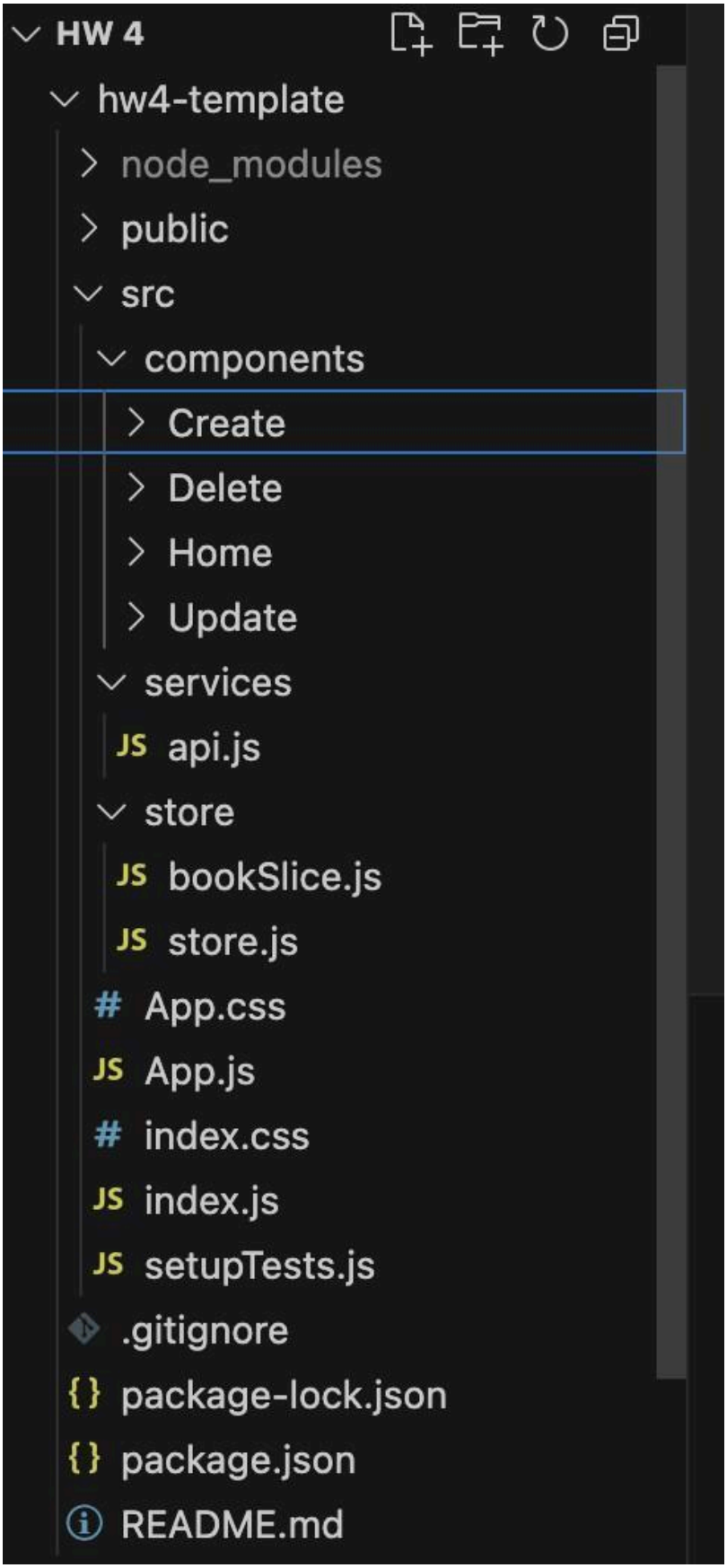

book_db
Tables
authors
Columns
id
first_name
last_name
email
created_at
updated_at
Indexes
PRIMARY
email
idx_email
Foreign Keys
Triggers

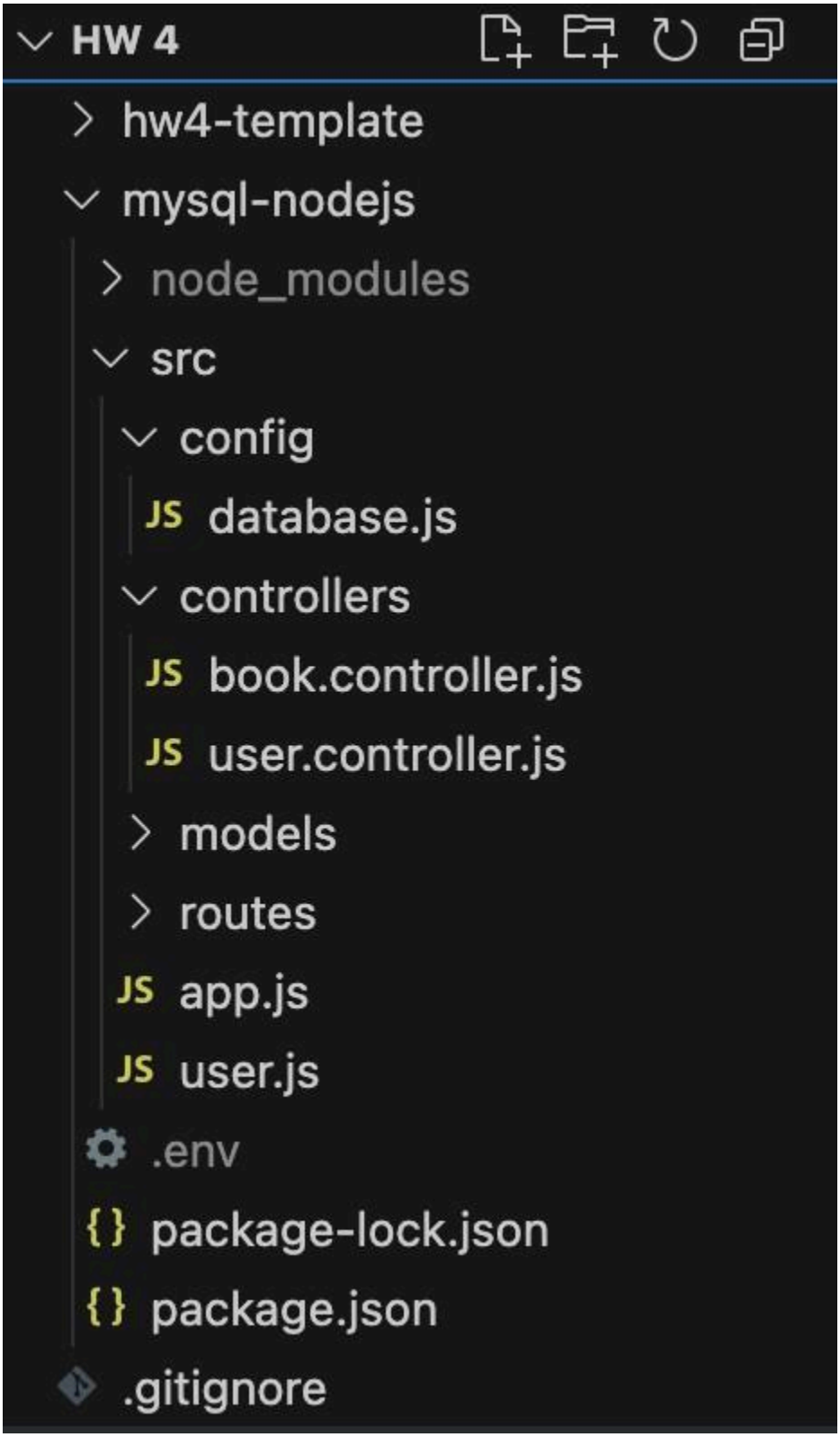

book_db
Tables
authors
books
Columns
id
title
isbn
publication_year
available_copies
author_id
created_at
updated_at
Indexes
PRIMARY
isbn
idx_isbn
idx_author_id
idx_title
Foreign Keys
books_ibfk_1

PART -2

I. Redux and Axios axis

- Used Axios to make the API calls.


HW 4
hw4-template
node_modules
public
src
components
Create
Delete
Home
Update
services
api.js
store
bookSlice.js
store.js
App.css
App.js
index.css
index.js
setupTests.js
.gitignore
package-lock.json
package.json
README.md


HW 4
hw4-template
mysql-nodejs
node_modules
src
config
database.js
controllers
book.controller.js
user.controller.js
models
routes
app.js
user.js
.env
package-lock.json
package.json
.gitignore

CreateBook.jsx | api.js | UpdateBook.jsx | DeleteBook.jsx

hw4-template > src > services > api.js

```
1   import axios from 'axios';
2
3   const api = axios.create({
4     baseURL: 'http://localhost:3000/api'
5   });
6
7   // Request interceptor for error handling
8   api.interceptors.request.use(
9     config => {
10      return config;
11    },
12    error => {
13      return Promise.reject(error);
14    }
15  );
16
17  // Response interceptor for error handling
18  api.interceptors.response.use(
19    response => response,
20    error => {
21      const message = error.response?.data?.message || 'An error occurred';
22      return Promise.reject(message);
23    }
24  );
25
26  // Get all books
27  export const getBooks = async () => {
28    const response = await api.get('/books');
29    return response.data;
30  };
31
32  // Get book by ID
33  export const getBookById = async (id) => {
34    const response = await api.get(`/books/${id}`);
35    return response.data;
36  };
37
38  // Create new book
39  export const createBook = async (bookData) => {
40    const response = await api.post('/books', bookData);
41    return response.data;
42  };
43
44  // Update existing book
45  export const updateBook = async (id, bookData) => {
46    const response = await api.put(`/books/${id}`, bookData);
47    return response.data;
48  };
49
50  // Delete book
51  export const deleteBook = async (id) => {
52    const response = await api.delete(`/books/${id}`);
53    return response.data;
54  };
55
56  export default api;
```

NOTE: I have used Reduxjs Toolkit

index.js | UpdateBook.jsx | DeleteBook.jsx

hw4-template > src > index.js

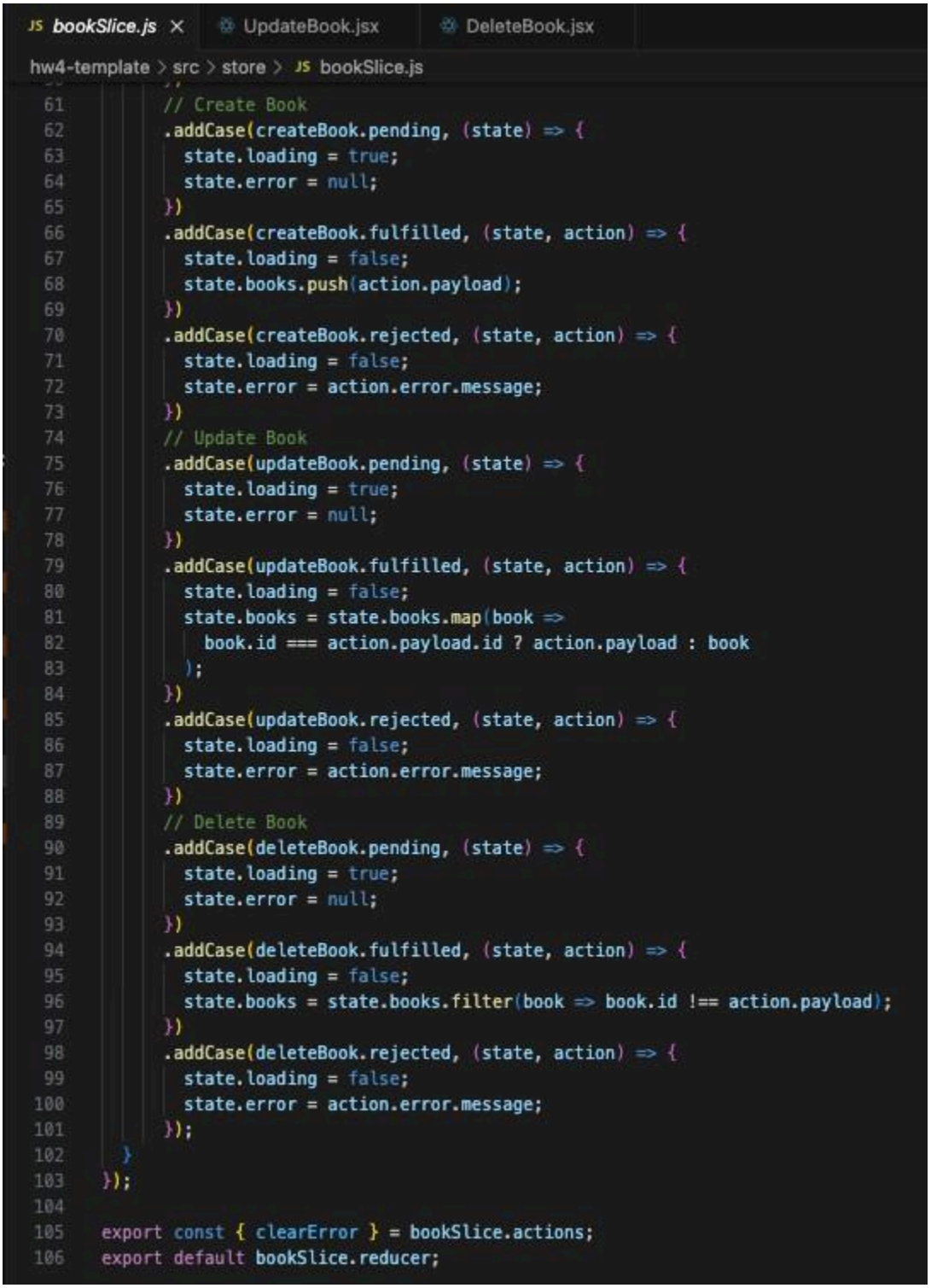
```
import React from 'react';
import ReactDOM from 'react-dom/client';
import { Provider } from 'react-redux';
import store from './store/store';
import App from './App';
import './index.css';

const root = ReactDOM.createRoot(document.getElementById('root'));
root.render(
  <React.StrictMode>
    <Provider store={store}>
      <App />
    </Provider>
  </React.StrictMode>
);
```

store.js | UpdateBook.jsx | DeleteBook.jsx

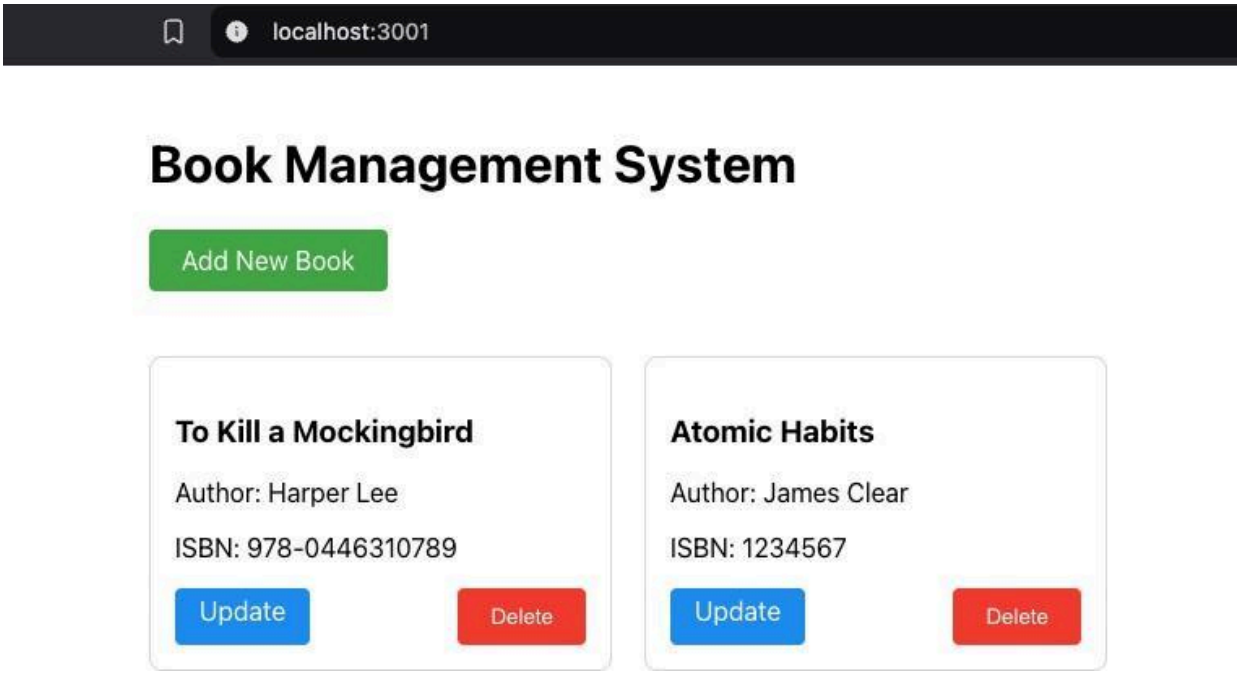
hw4-template > src > store > store.js

```
import { configureStore } from '@reduxjs/toolkit';
import bookReducer from './bookSlice';

const store = configureStore({
  reducer: {
    books: bookReducer
  },
  middleware: (getDefaultMiddleware) =>
    getDefaultMiddleware({
      serializableCheck: false
    }),
  devTools: process.env.NODE_ENV !== 'production'
});

export default store;
```


bookSlice.js
UpdateBook.jsx
DeleteBook.jsx
hw4-template > src > store > bookSlice.js
import { createSlice, createAsyncThunk } from '@reduxjs/toolkit
import * as api from '../services/api';
// Async Thunks
export const fetchBooks = createAsyncThunk(
'books/fetchBooks',
async () => {
return await api.getBooks();
}
);
export const createBook = createAsyncThunk(
'books/createBook',
async (bookData) => {
return await api.createBook(bookData);
}
);
export const updateBook = createAsyncThunk(
'books/updateBook',
async ({ id, bookData }) => {
return await api.updateBook(id, bookData);
}
);
export const deleteBook = createAsyncThunk(
'books/deleteBook',
async (id) => {
await api.deleteBook(id);
return id;
}
);
const bookSlice = createSlice({
name: 'books',
initialState: {
books: [],
loading: false,
error: null
},
reducers: {
clearError: (state) => {
state.error = null;
}
},
extraReducers: (builder) => {
builder
// Fetch Books
.addCase(fetchBooks.pending, (state) => {
state.loading = true;
state.error = null;
})
.addCase(fetchBooks.fulfilled, (state, action) => {
state.loading = false;
state.books = action.payload;
})
.addCase(fetchBooks.rejected, (state, action) => {
state.loading = false;
state.error = action.error.message;
})


bookSlice.js
UpdateBook.jsx
DeleteBook.jsx
hw4-template > src > store > bookSlice.js
// Create Book
.addCase(createBook.pending, (state) => {
state.loading = true;
state.error = null;
})
.addCase(createBook.fulfilled, (state, action) => {
state.loading = false;
state.books.push(action.payload);
})
.addCase(createBook.rejected, (state, action) => {
state.loading = false;
state.error = action.error.message;
})
// Update Book
.addCase(updateBook.pending, (state) => {
state.loading = true;
state.error = null;
})
.addCase(updateBook.fulfilled, (state, action) => {
state.loading = false;
state.books = state.books.map(book =>
book.id === action.payload.id ? action.payload : book
);
})
.addCase(updateBook.rejected, (state, action) => {
state.loading = false;
state.error = action.error.message;
})
// Delete Book
.addCase(deleteBook.pending, (state) => {
state.loading = true;
state.error = null;
})
.addCase(deleteBook.fulfilled, (state, action) => {
state.loading = false;
state.books = state.books.filter(book => book.id !== action.payload);
})
.addCase(deleteBook.rejected, (state, action) => {
state.loading = false;
state.error = action.error.message;
});
}
});
export const { clearError } = bookSlice.actions;
export default bookSlice.reducer;

Home Screen

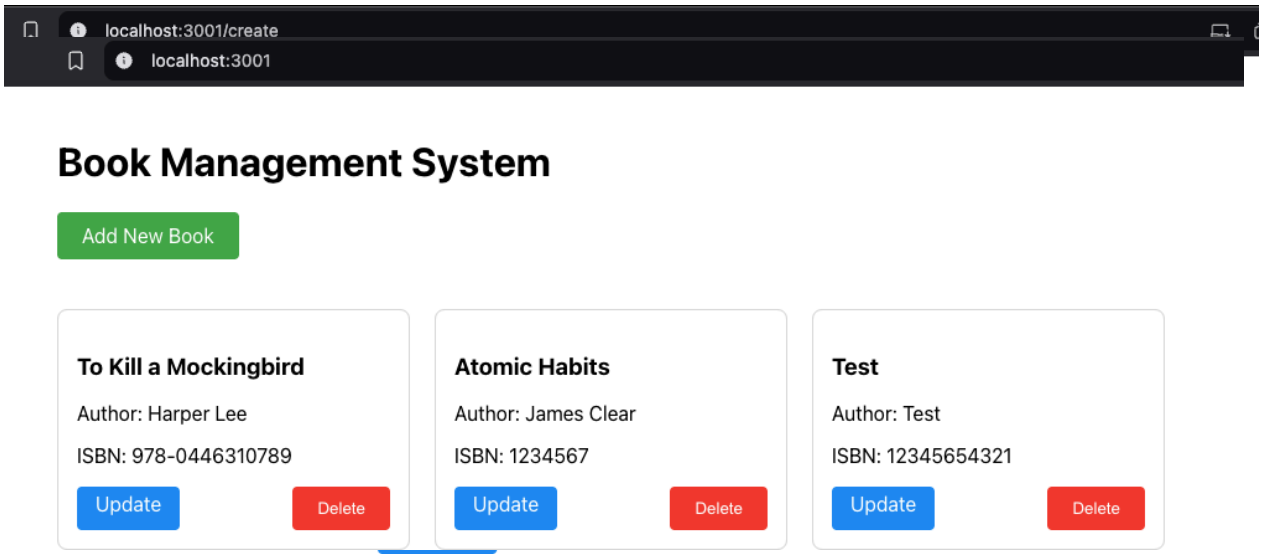

localhost:3001
Book Management System
Add New Book
To Kill a Mockingbird
Author: Harper Lee
ISBN: 978-0446310789
Update
Delete
Atomic Habits
Author: James Clear
ISBN: 1234567
Update
Delete

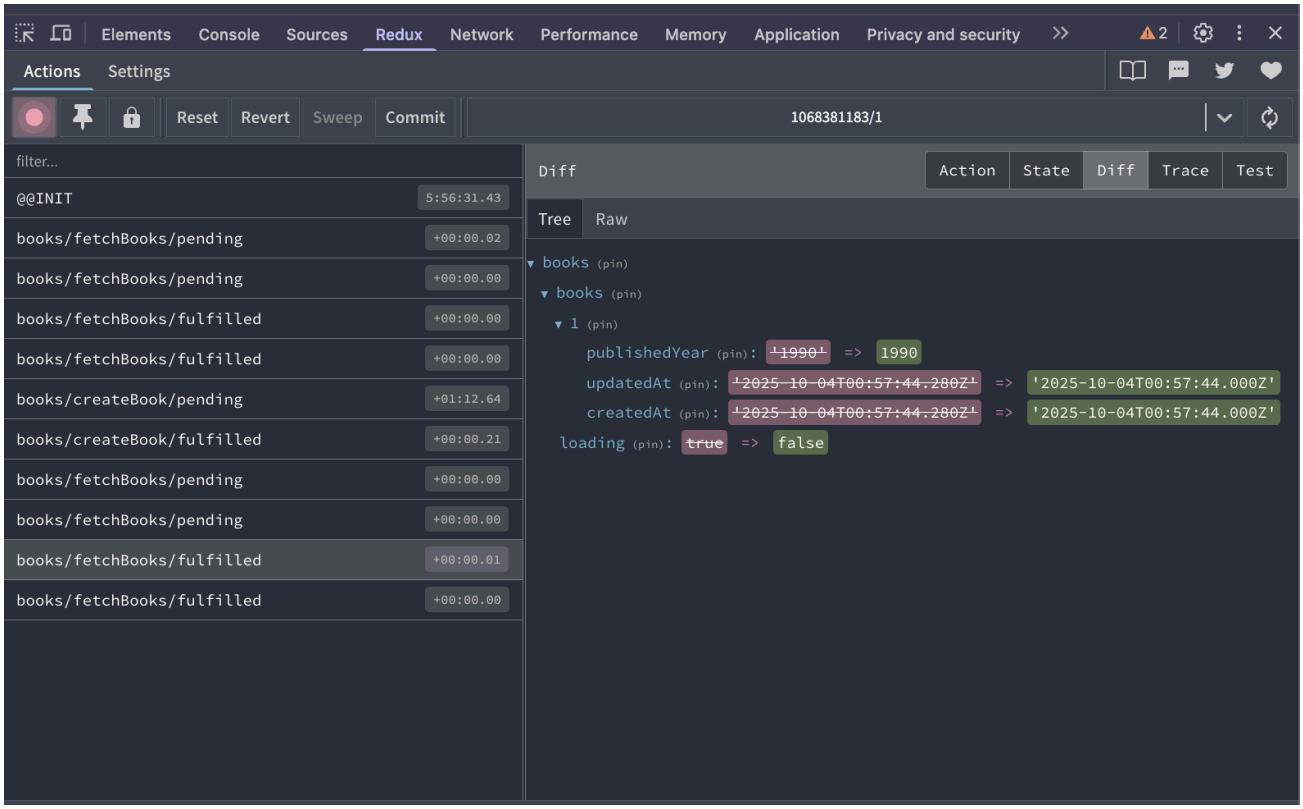
Home.jsx | UpdateBook.jsx | DeleteBook.jsx

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

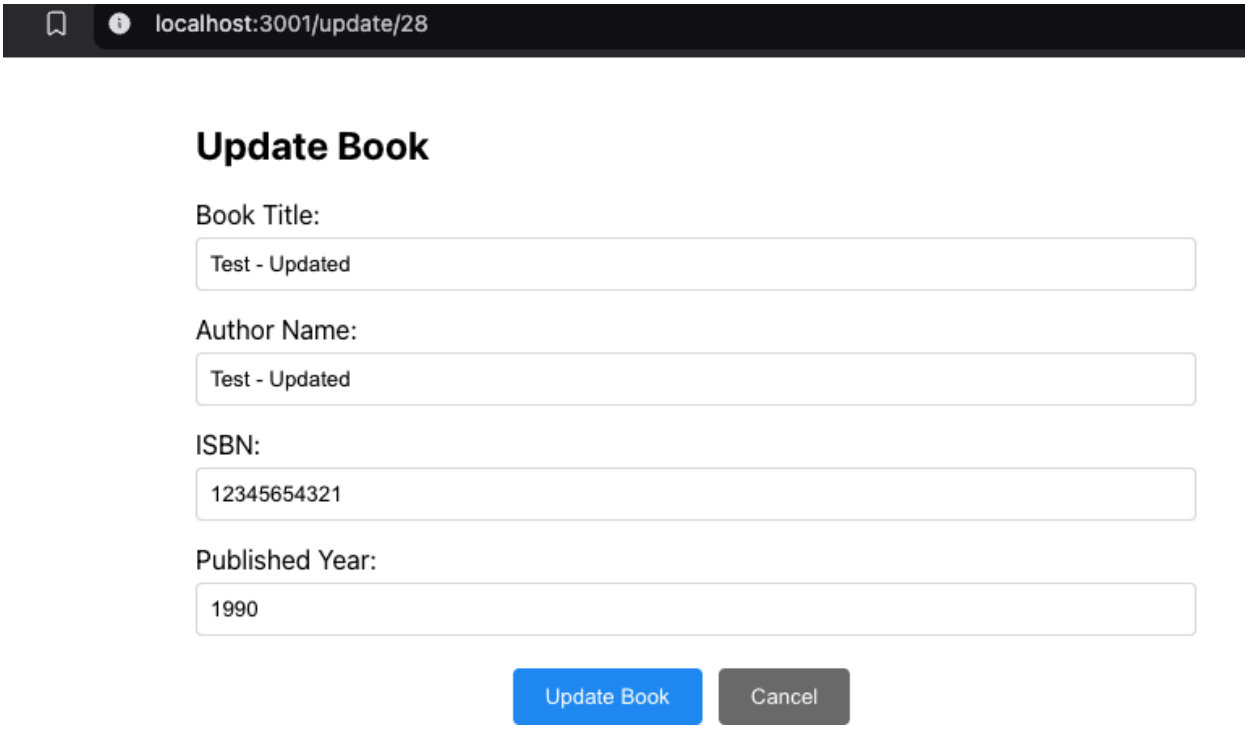
```jsx
import React, { useEffect } from 'react';
import { Link } from 'react-router-dom';
import { useDispatch, useSelector } from 'react-redux';
import { fetchBooks, deleteBook } from '../../store/bookSlice';
import './Home.css';

const Home = () => {
  const dispatch = useDispatch();
  const { books, loading, error } = useSelector(state => state.books);

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

  const handleDelete = async (id) => {
    try {
      await dispatch(deleteBook(id)).unwrap();
    } catch (err) {
      console.error('Failed to delete book:', err);
    }
  };

  if (loading) return <div>Loading...</div>;
  if (error) return <div>Error: {error}</div>;

  return (
    <div className="home-container">
      <h1>Book Management System</h1>
      <Link to="/create" className="add-button">Add New Book</Link>

      <div className="books-grid">
        {books.map(book => (
          <div key={book.id} className="book-card">
            <h3>{book.title}</h3>
            <p>Author: {book.author}</p>
            <p>ISBN: {book.isbn}</p>
            <div className="book-actions">
              <Link to={`/update/${book.id}`} className="edit-button">Update</Link>
              <button
                onClick={() => handleDelete(book.id)}
                className="delete-button"
              >
                Delete
              </button>
            </div>
          </div>
        ))}
      </div>
    </div>
  );
};

export default Home;
```

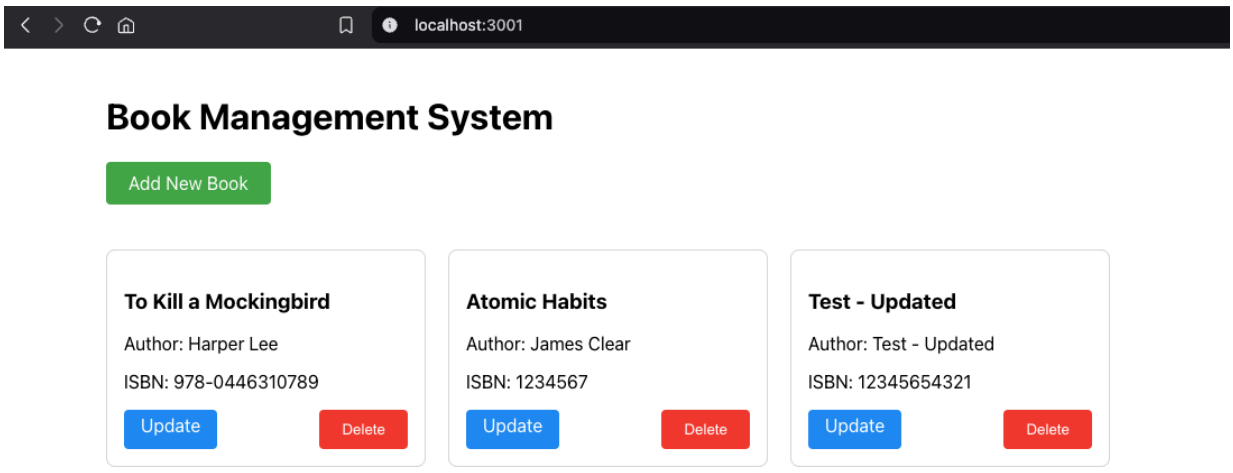
Create Screen

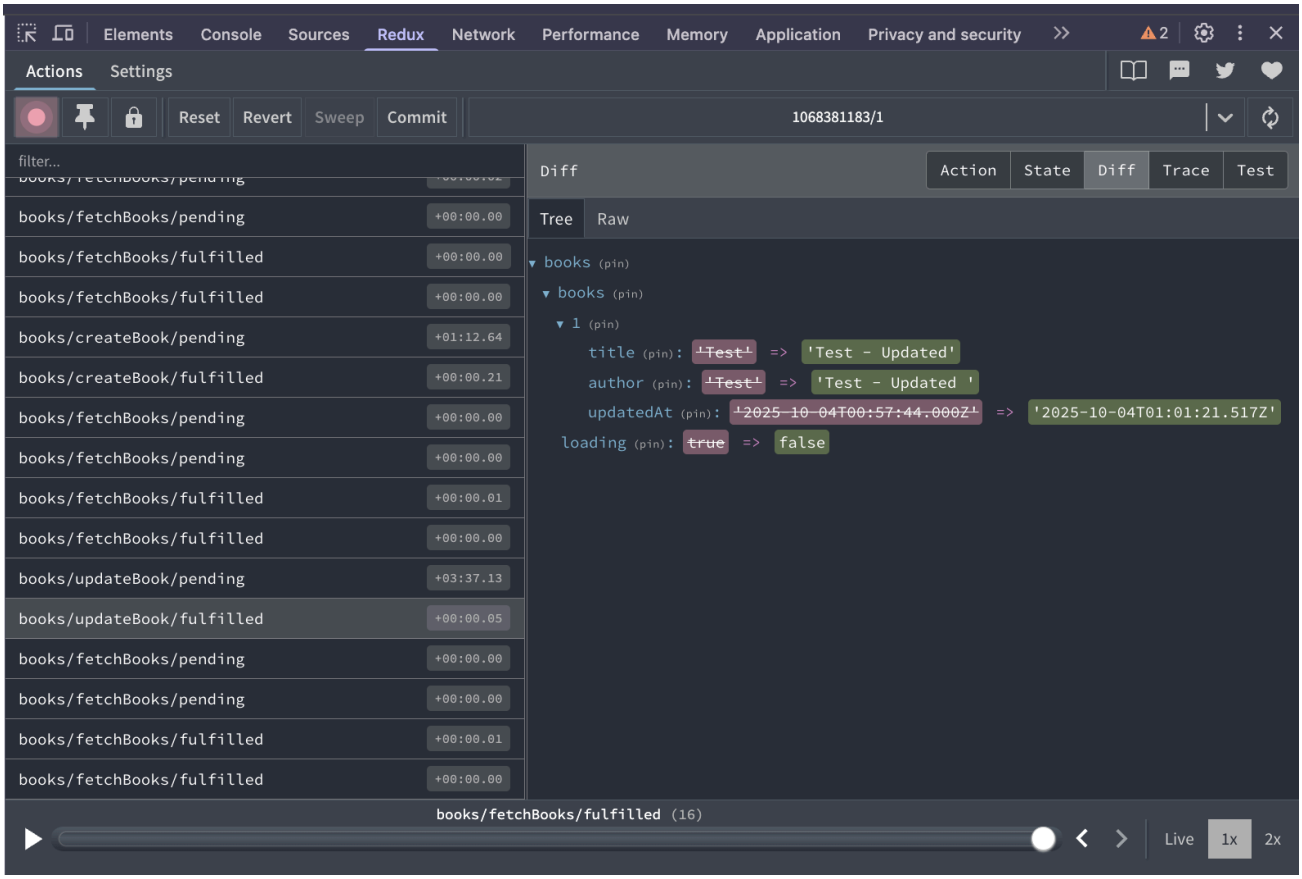


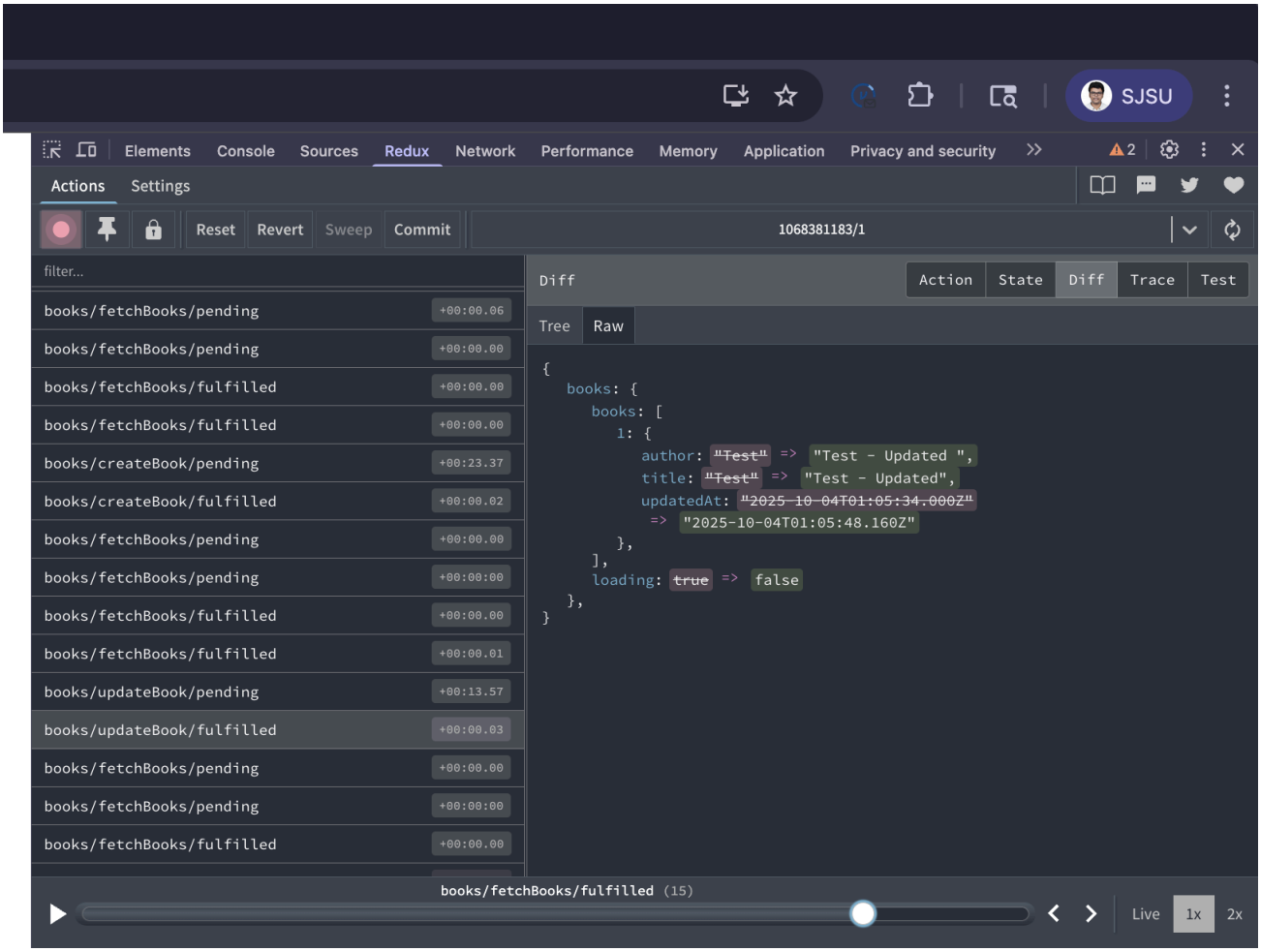

Elements
Console
Sources
Redux
Network
Performance
Memory
Application
Privacy and security
2
Actions
Settings
Reset
Revert
Sweep
Commit
1068381183/1
filter...
@@INIT 5:56:31.43
books/fetchBooks/pending +00:00.02
books/fetchBooks/pending +00:00.00
books/fetchBooks/fulfilled +00:00.00
books/fetchBooks/fulfilled +00:00.00
books/createBook/pending +01:12.64
books/createBook/fulfilled +00:00.21
books/fetchBooks/pending +00:00.00
books/fetchBooks/pending +00:00.00
books/fetchBooks/fulfilled +00:00.01
books/fetchBooks/fulfilled +00:00.00
Diff
Action
State
Diff
Trace
Test
Tree
Raw
books (pin)
books (pin)
1 (pin)
publishedYear (pin): 1990 => 1990
updatedAt (pin): '2025-10-04T00:57:44.280Z' => '2025-10-04T00:57:44.000Z'
createdAt (pin): '2025-10-04T00:57:44.280Z' => '2025-10-04T00:57:44.000Z'
loading (pin): true => false

Update Screen


localhost:3001/update/28
Update Book
Book Title:
Test - Updated
Author Name:
Test - Updated
ISBN:
12345654321
Published Year:
1990
Update Book
Cancel


localhost:3001
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 - Updated
Author: Test - Updated
ISBN: 12345654321
Update
Delete


Elements
Console
Sources
Redux
Network
Performance
Memory
Application
Privacy and security
Actions
Settings
Reset
Revert
Sweep
Commit
1068381183/1
filter...
books/fetchBooks/pending +00:00.00
books/fetchBooks/fulfilled +00:00.00
books/fetchBooks/fulfilled +00:00.00
books/createBook/pending +01:12.64
books/createBook/fulfilled +00:00.21
books/fetchBooks/pending +00:00.00
books/fetchBooks/pending +00:00.00
books/fetchBooks/fulfilled +00:00.01
books/fetchBooks/fulfilled +00:00.00
books/updateBook/pending +03:37.13
books/updateBook/fulfilled +00:00.05
books/fetchBooks/pending +00:00.00
books/fetchBooks/pending +00:00.00
books/fetchBooks/fulfilled +00:00.01
books/fetchBooks/fulfilled +00:00.00
Diff
Action
State
Diff
Trace
Test
Tree
Raw
books (pin)
books (pin)
1 (pin)
title (pin): 'Test' => 'Test - Updated'
author (pin): 'Test' => 'Test - Updated '
updatedAt (pin): '2025-10-04T00:57:44.000Z' => '2025-10-04T01:01:21.517Z'
loading (pin): true => false
books/fetchBooks/fulfilled (16)
Live
1x
2x

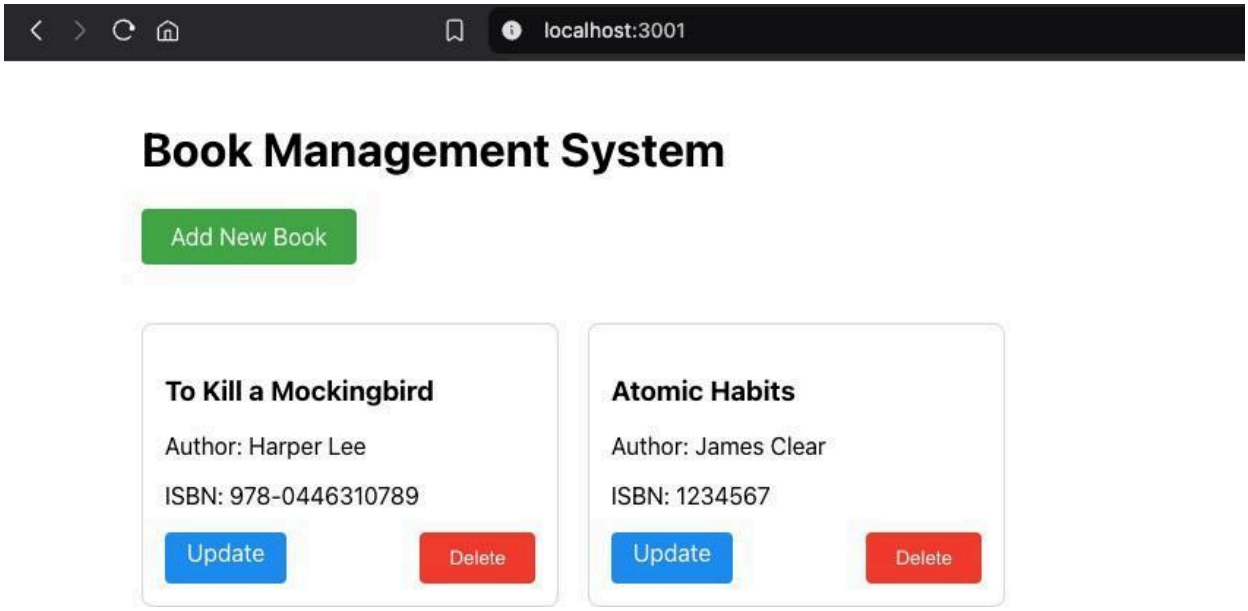

SJSU
Elements
Console
Sources
Redux
Network
Performance
Memory
Application
Privacy and security
Actions
Settings
Reset
Revert
Sweep
Commit
1068381183/1
filter...
books/fetchBooks/pending +00:00.06
books/fetchBooks/pending +00:00.00
books/fetchBooks/fulfilled +00:00.00
books/fetchBooks/fulfilled +00:00.00
books/createBook/pending +00:23.37
books/createBook/fulfilled +00:00.02
books/fetchBooks/pending +00:00.00
books/fetchBooks/pending +00:00:00
books/fetchBooks/fulfilled +00:00.00
books/fetchBooks/fulfilled +00:00.01
books/updateBook/pending +00:13.57
books/updateBook/fulfilled +00:00.03
books/fetchBooks/pending +00:00.00
books/fetchBooks/pending +00:00:00
books/fetchBooks/fulfilled +00:00.00
Diff
Action
State
Diff
Trace
Test
Tree
Raw
{
books: {
books: [
1: {
author: "Test" => "Test - Updated ",
title: "Test" => "Test - Updated",
updatedAt: "2025-10-04T01:05:34.000Z"
=> "2025-10-04T01:05:48.160Z"
},
],
loading: true => false
},
}
books/fetchBooks/fulfilled (15)
Live
1x
2x

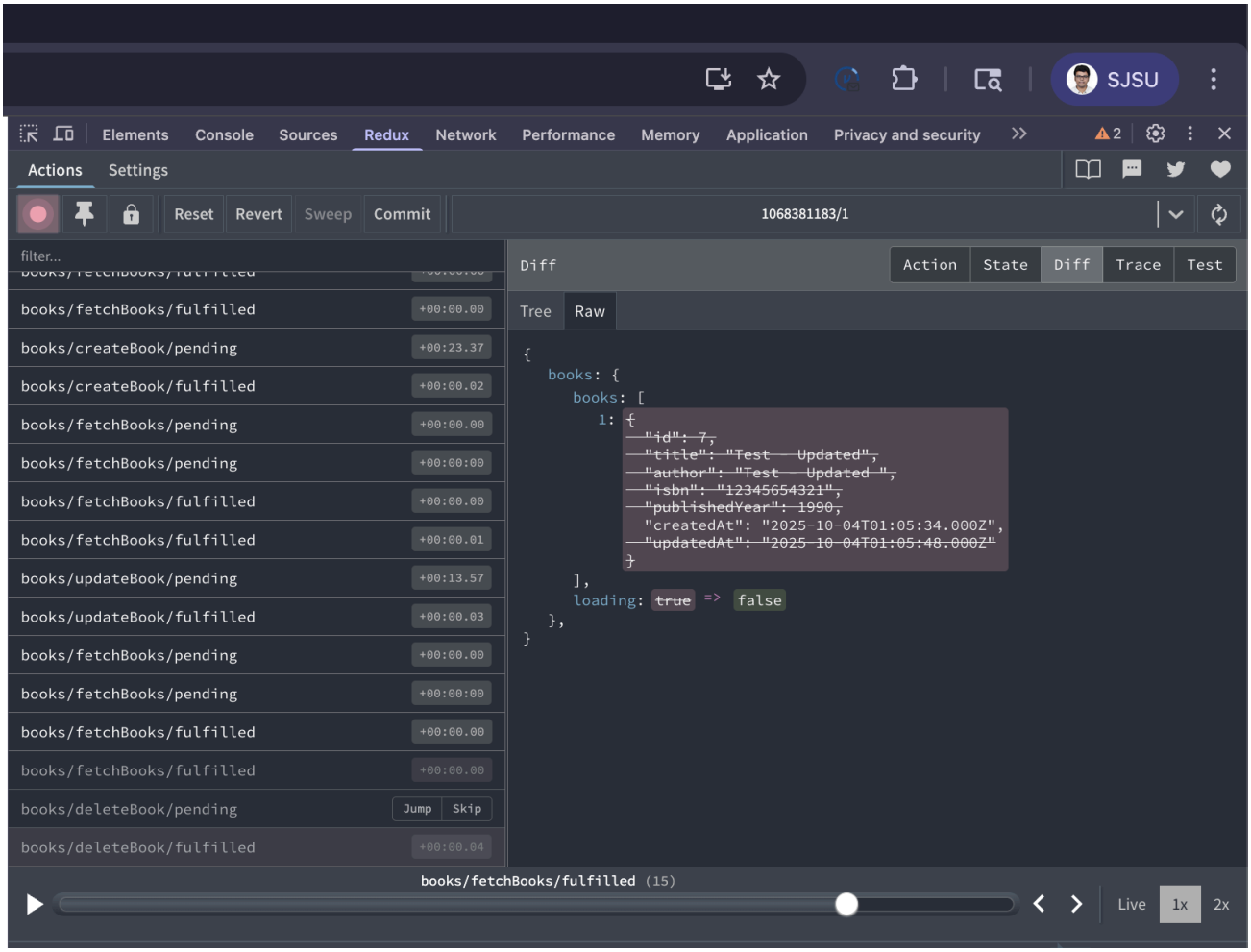
CreateBook.jsx | UpdateBook.jsx | DeleteBook.jsx

hw4-template > src > components > Update > UpdateBook.jsx

```jsx
import React, { useState, useEffect } from 'react';
import { useNavigate, useParams } from 'react-router-dom';
import { useDispatch, useSelector } from 'react-redux';
import { updateBook } from '../../store/bookSlice';
import { getBookById } from '../../services/api';
import './UpdateBook.css';

const UpdateBook = () => {
  const [formData, setFormData] = useState({
    title: '',
    author: '',
    isbn: '',
    publishedYear: ''
  });

  const navigate = useNavigate();
  const dispatch = useDispatch();
  const { id } = useParams();
  const { loading, error } = useSelector(state => state.books);
  const [fetchError, setFetchError] = useState(null);

  useEffect(() => {
    const fetchBook = async () => {
      try {
        const book = await getBookById(id);
        setFormData(book);
      } catch (error) {
        setFetchError(error.message || 'Failed to fetch book');
      }
    };

    fetchBook();
  }, [id]);

  const handleChange = (e) => {
    const { name, value } = e.target;
    setFormData(prev => ({
      ...prev,
      [name]: value
    }));
  };

  const handleSubmit = async (e) => {
    e.preventDefault();
    try {
      await dispatch(updateBook({ id, bookData: formData })).unwrap();
      navigate('/');
    } catch (err) {
      console.error('Failed to update book:', err);
    }
  };

  if (loading) return <div>Loading...</div>;
  if (fetchError) return <div>Error: {fetchError}</div>;

  return (
    <div className="update-book-container">
      <h2>Update Book</h2>
      {error && <div className="error-message">{error}</div>}
      <form onSubmit={handleSubmit}>
        <div className="form-group">
          <label htmlFor="title">Book Title:</label>
```

Delete Screen




SJSU
Elements
Console
Sources
Redux
Network
Performance
Memory
Application
Privacy and security
Actions
Settings
Reset
Revert
Sweep
Commit
1068381183/1
filter...
books/fetchBooks/fulfilled +00:00.00
books/createBook/pending +00:23.37
books/createBook/fulfilled +00:00.02
books/fetchBooks/pending +00:00.00
books/fetchBooks/pending +00:00:00
books/fetchBooks/fulfilled +00:00.00
books/fetchBooks/fulfilled +00:00.01
books/updateBook/pending +00:13.57
books/updateBook/fulfilled +00:00.03
books/fetchBooks/pending +00:00.00
books/fetchBooks/pending +00:00:00
books/fetchBooks/fulfilled +00:00.00
books/fetchBooks/fulfilled +00:00.00
books/deleteBook/pending Jump Skip
books/deleteBook/fulfilled +00:00.04
Diff
Action
State
Diff
Trace
Test
Tree
Raw
{
books: {
books: [
1: {
"id": 7,
"title": "Test - Updated",
"author": "Test - Updated ",
"isbn": "12345654321",
"publishedYear": 1990,
"createdAt": "2025-10-04T01:05:34.000Z",
"updatedAt": "2025-10-04T01:05:48.000Z"
}
],
loading: true => false
},
}
books/fetchBooks/fulfilled (15)
Live
1x
2x

CreateBook.jsx | UpdateBook.jsx | DeleteBook.jsx

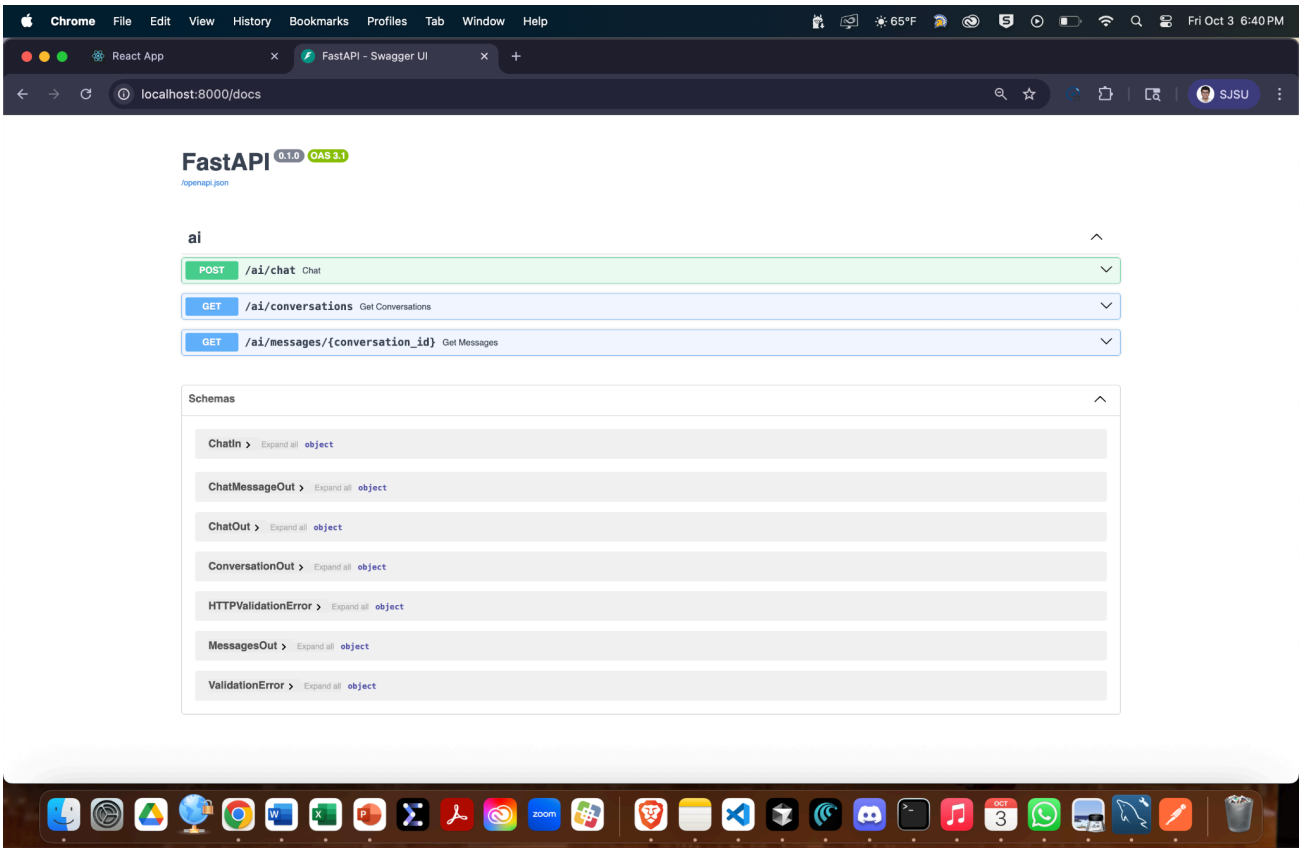
hw4-template > src > components > Delete > DeleteBook.jsx

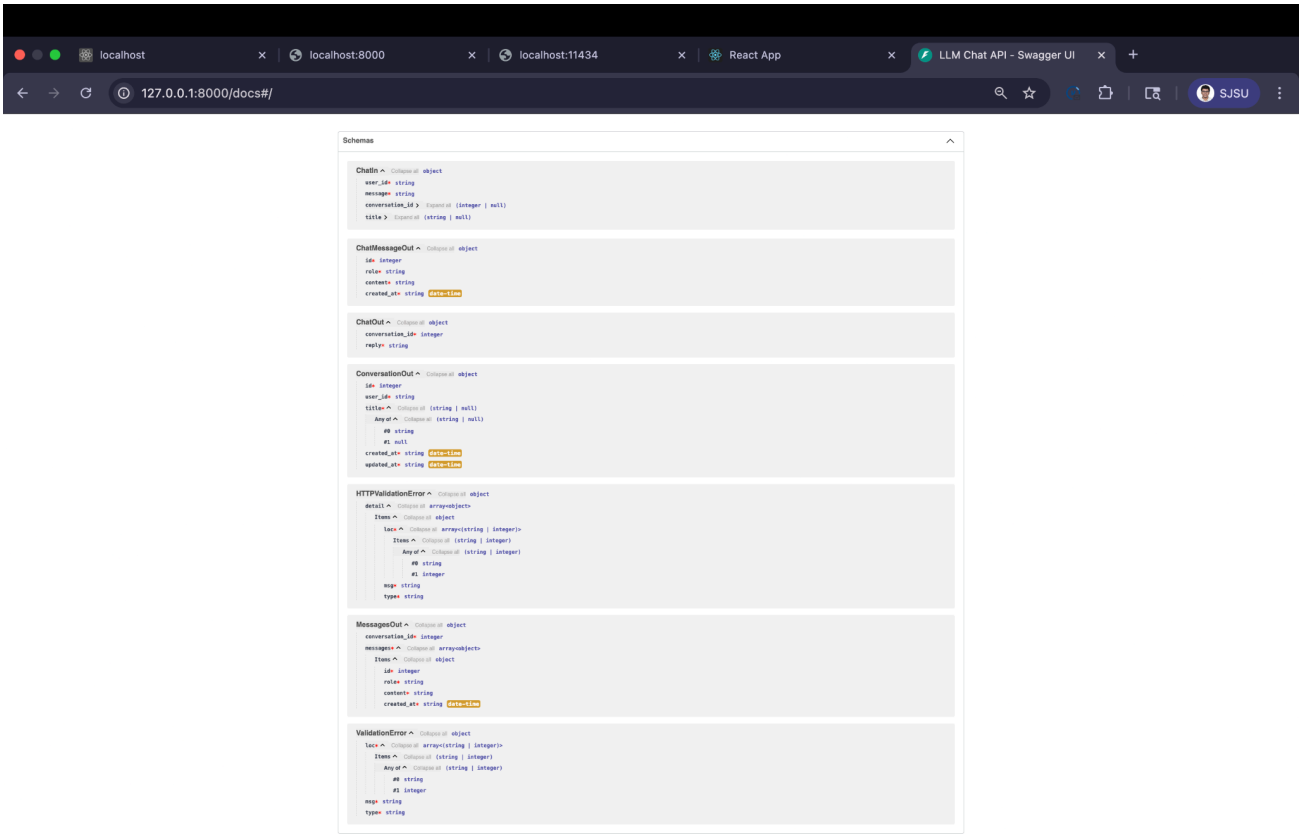
```jsx
import React, { useState, useEffect } from 'react';
import { useNavigate, useParams } from 'react-router-dom';
import { useDispatch, useSelector } from 'react-redux';
import { deleteBook } from '../../store/bookSlice';
import { getBookById } from '../../services/api';
import './DeleteBook.css';

const DeleteBook = () => {
  const [book, setBook] = useState(null);
  const [error, setError] = useState(null);
  const navigate = useNavigate();
  const dispatch = useDispatch();
  const { id } = useParams();
  const { loading } = useSelector(state => state.books);

  useEffect(() => {
    const fetchBook = async () => {
      try {
        const bookData = await getBookById(id);
        setBook(bookData);
      } catch (error) {
        setError('Failed to fetch book details');
        console.error('Error:', error);
      }
    };

    fetchBook();
  }, [id]);

  const handleDelete = async () => {
    try {
      await dispatch(deleteBook(id)).unwrap();
      navigate('/');
    } catch (error) {
      setError(error.message);
      console.error('Error:', error);
    }
  };

  if (loading) return <div>Loading...</div>;
  if (error) return <div className="error-message">{error}</div>;

  return (
    <div className="delete-book-container">
      <h2>Delete Book</h2>
      {book && (
        <div className="book-details">
          <h3>{book.title}</h3>
          <p>Author: {book.author}</p>
          <p>ISBN: {book.isbn}</p>
          {book.publishedYear && <p>Published Year: {book.publishedYear}</p>}
          <p className="warning-text">Are you sure you want to delete this book?</p>
        </div>
      )}
```

PART -3

LLM Chat (Ollama + FastAPI + React)

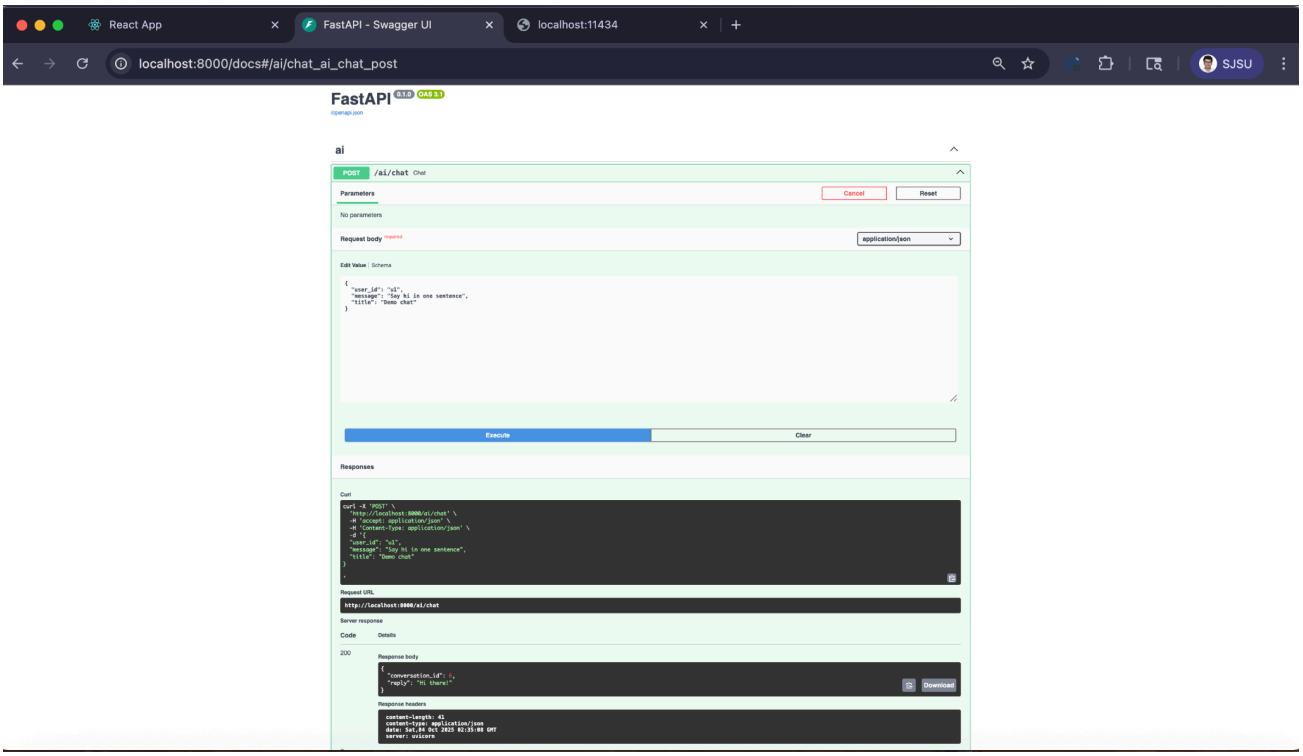
API responses in Swagger/Postman:



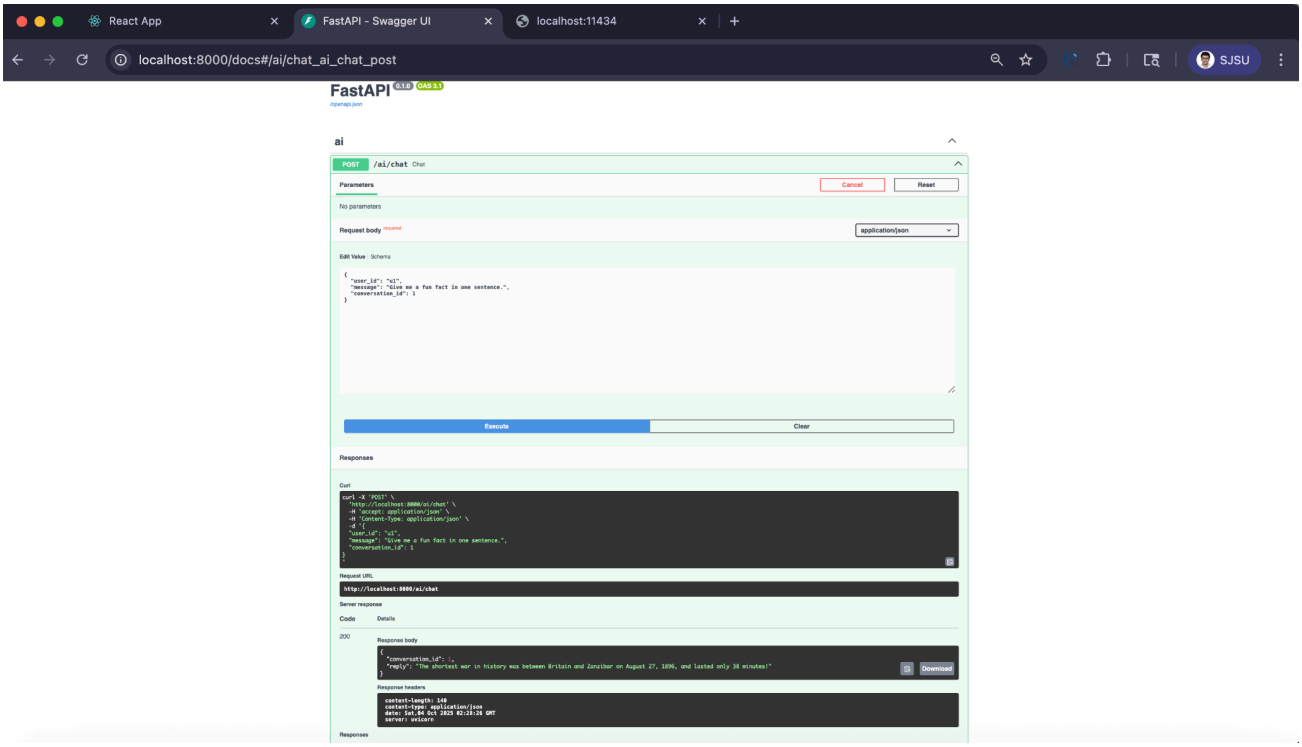

localhost
localhost:8000
localhost:11434
React App
LLM Chat API - Swagger UI
127.0.0.1:8000/docs#/
SJSU
Schemas
ChatIn
ChatMessageOut
ChatOut
ConversationOut
HTTPValidationError
MessagesOut
ValidationError

1) POST /ai/chat — start a conversation

Swagger:

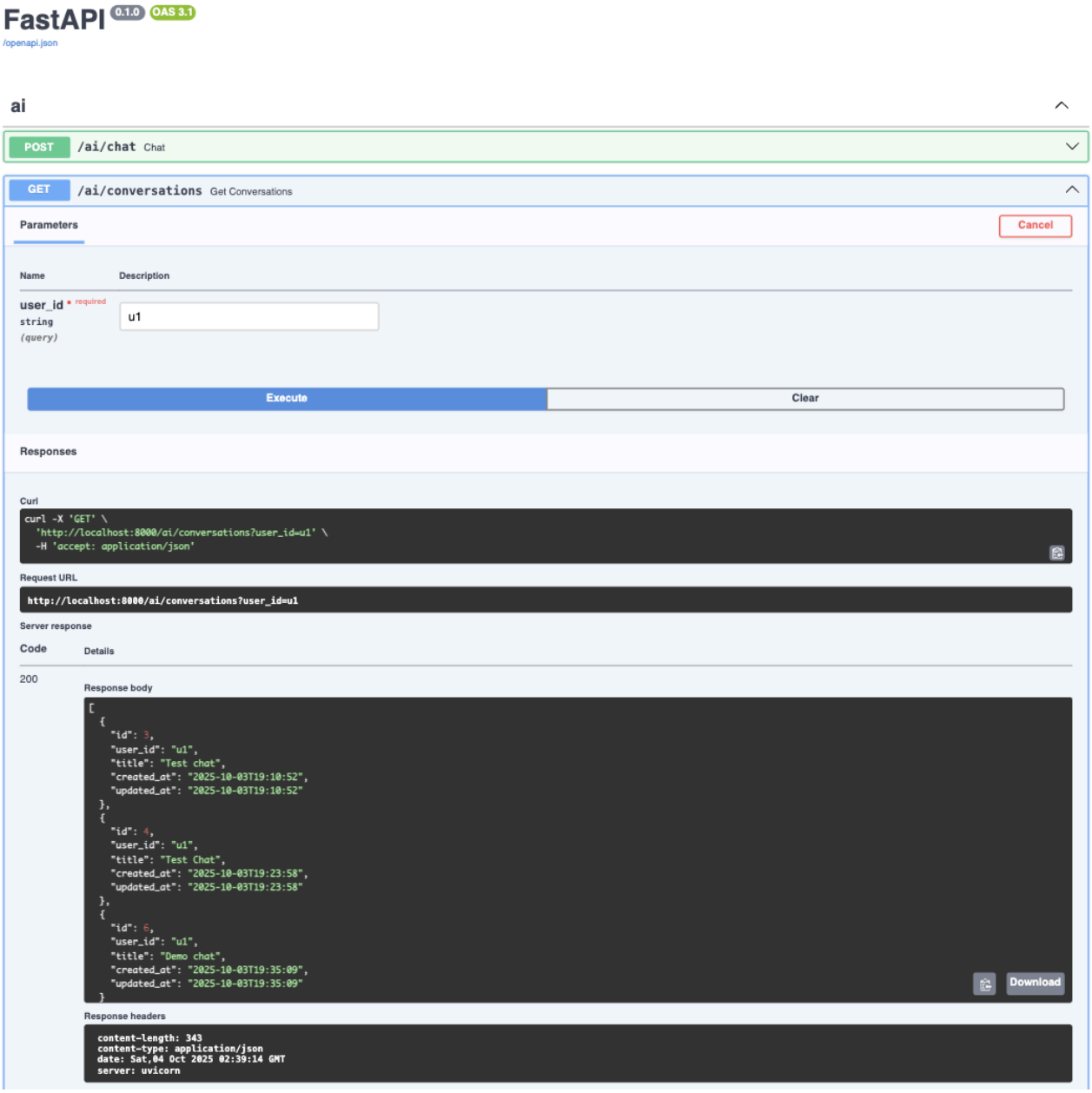


2) POST /ai/chat — continue the same conversation


React App
FastAPI - Swagger UI
localhost:11434
localhost:8000/docs#/ai/chat_ai_chat_post
SJSU
FastAPI
0.1.0
OAS 3.1
/openapi.json
ai
POST
/ai/chat
Chat
Parameters
Cancel
Reset
No parameters
Request body
required
application/json
Edit Value
Schema
{
"user_id": "u1",
"message": "Give me a fun fact in one sentence.",
"conversation_id": 1
}
Execute
Clear
Responses
Curl
curl -X 'POST' \
'http://localhost:8000/ai/chat' \
-H 'accept: application/json' \
-H 'Content-Type: application/json' \
-d '{
"user_id": "u1",
"message": "Give me a fun fact in one sentence.",
"conversation_id": 1
}'
Request URL
http://localhost:8000/ai/chat
Server response
Code
Details
200
Response body
{
"conversation_id": ,
"reply": "The shortest war in history was between Britain and Zanzibar on August 27, 1896, and lasted only 38 minutes!"
}
Download
Response headers
content-length: 148
content-type: application/json
date: Sat,04 Oct 2025 02:28:28 GMT
server: uvicorn
Responses

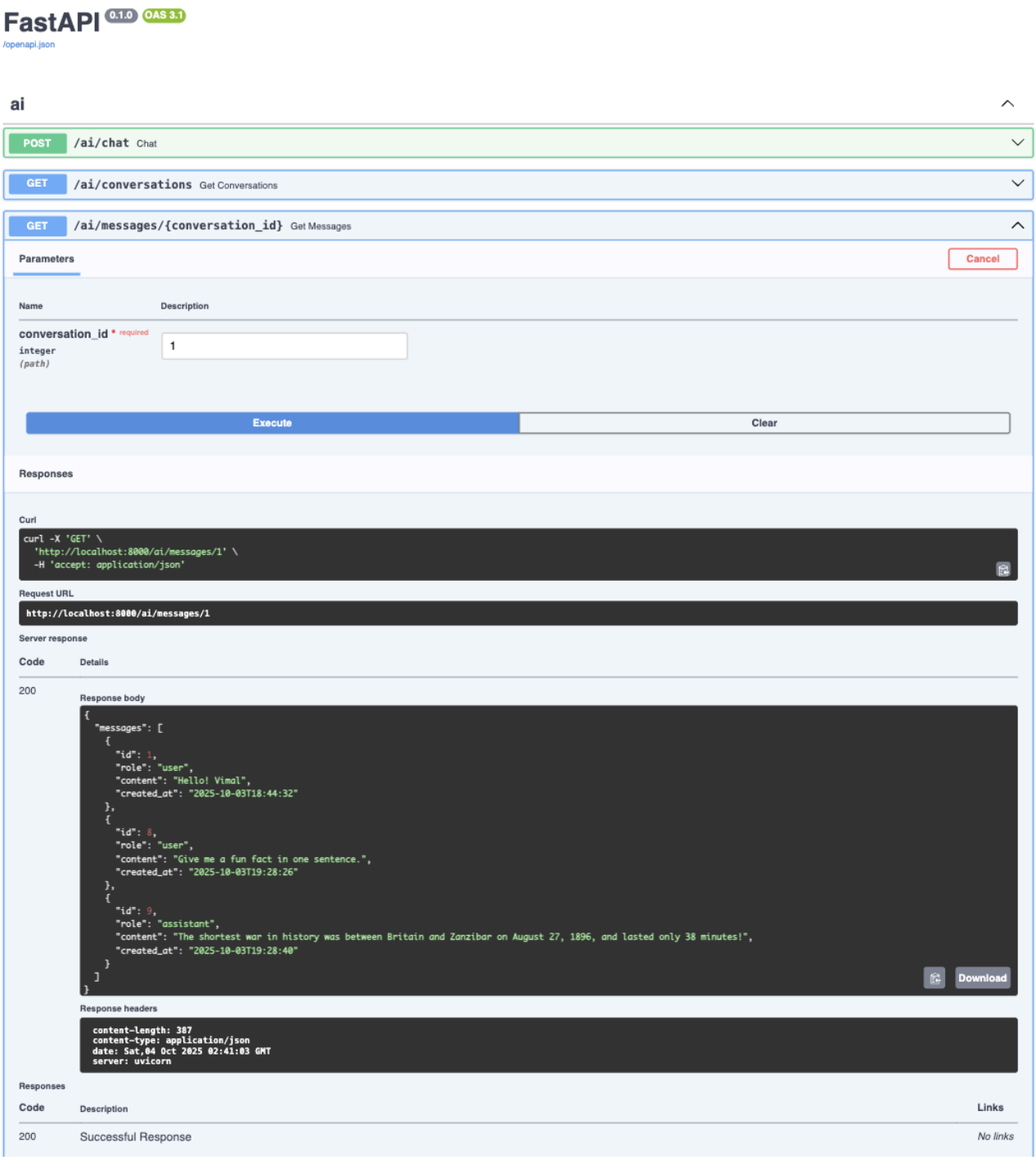
3) GET /ai/conversations — list the user's chats

Swagger: set user_id = u1 in the query param



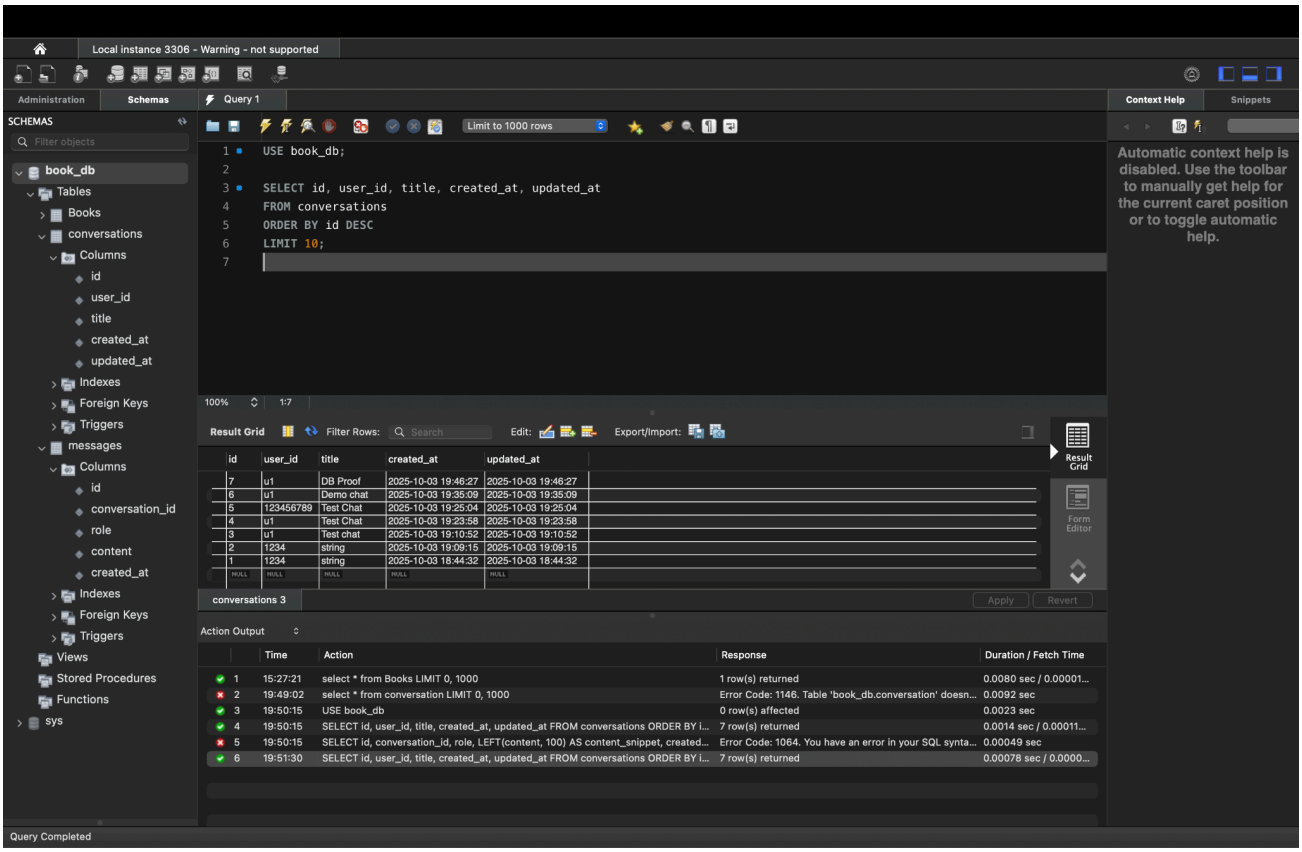
4) GET /ai/messages/{conversation_id} — all messages in a chat

Swagger: path param conversation_id = 1

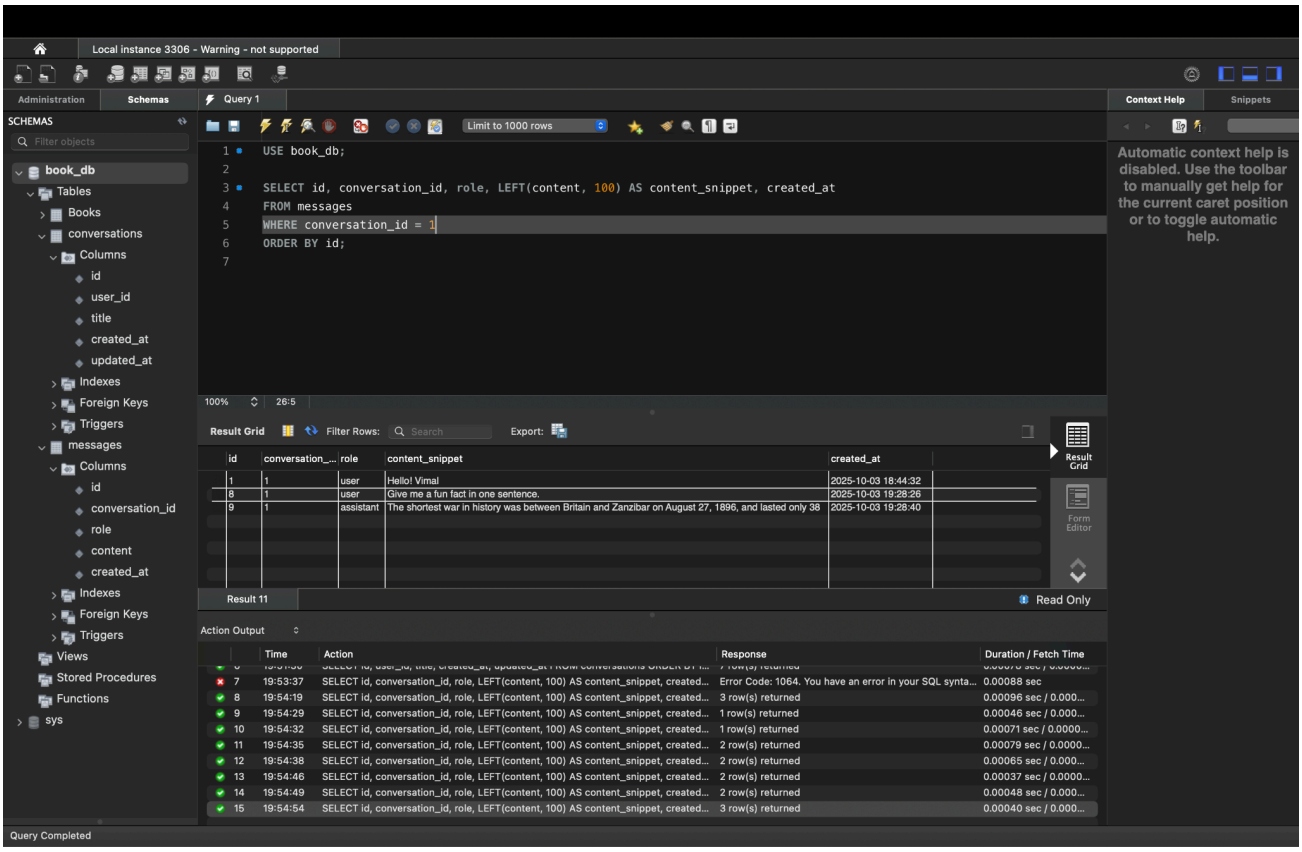


DB Screenshots:

Conversations list



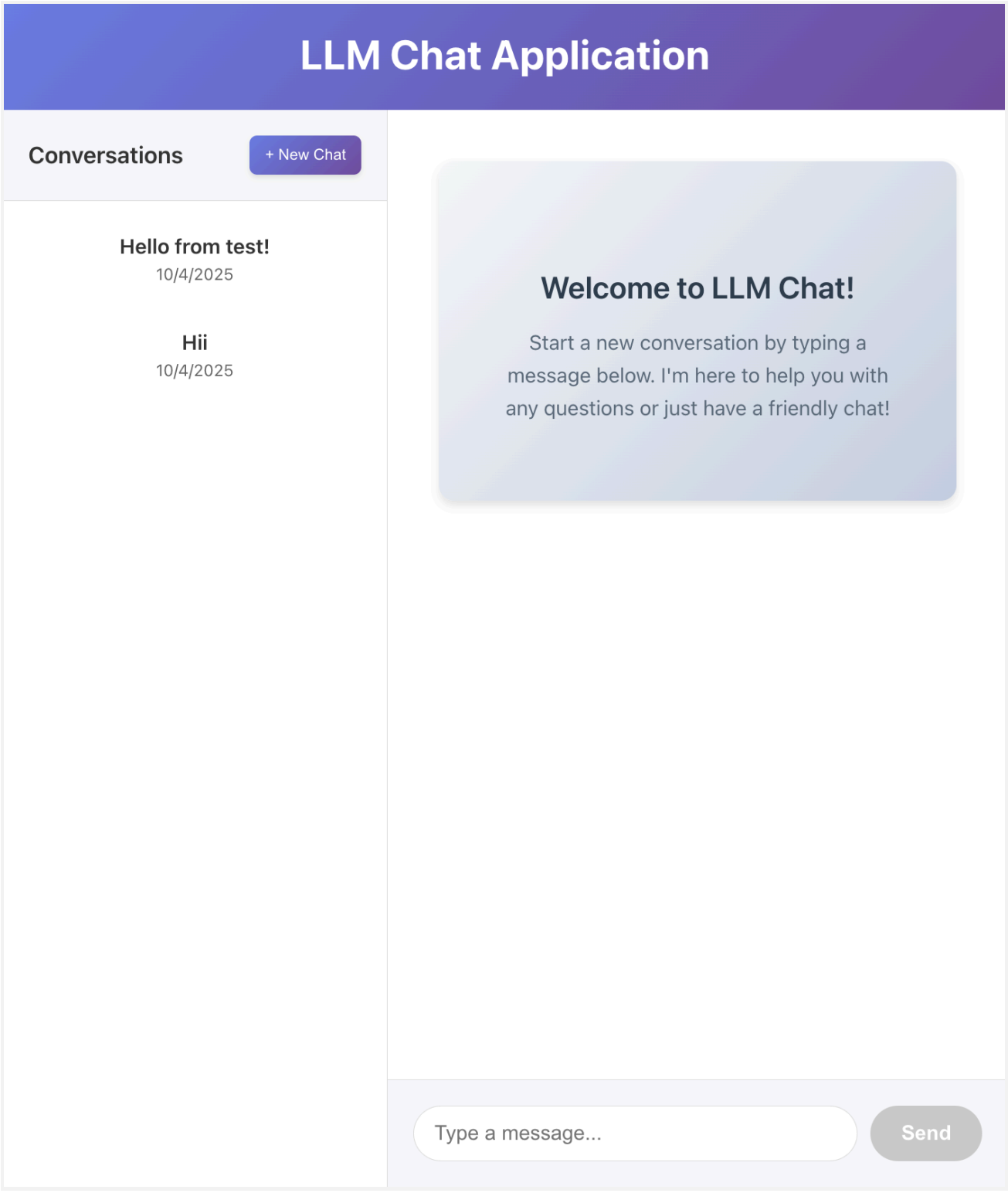
Messages for one conversation :

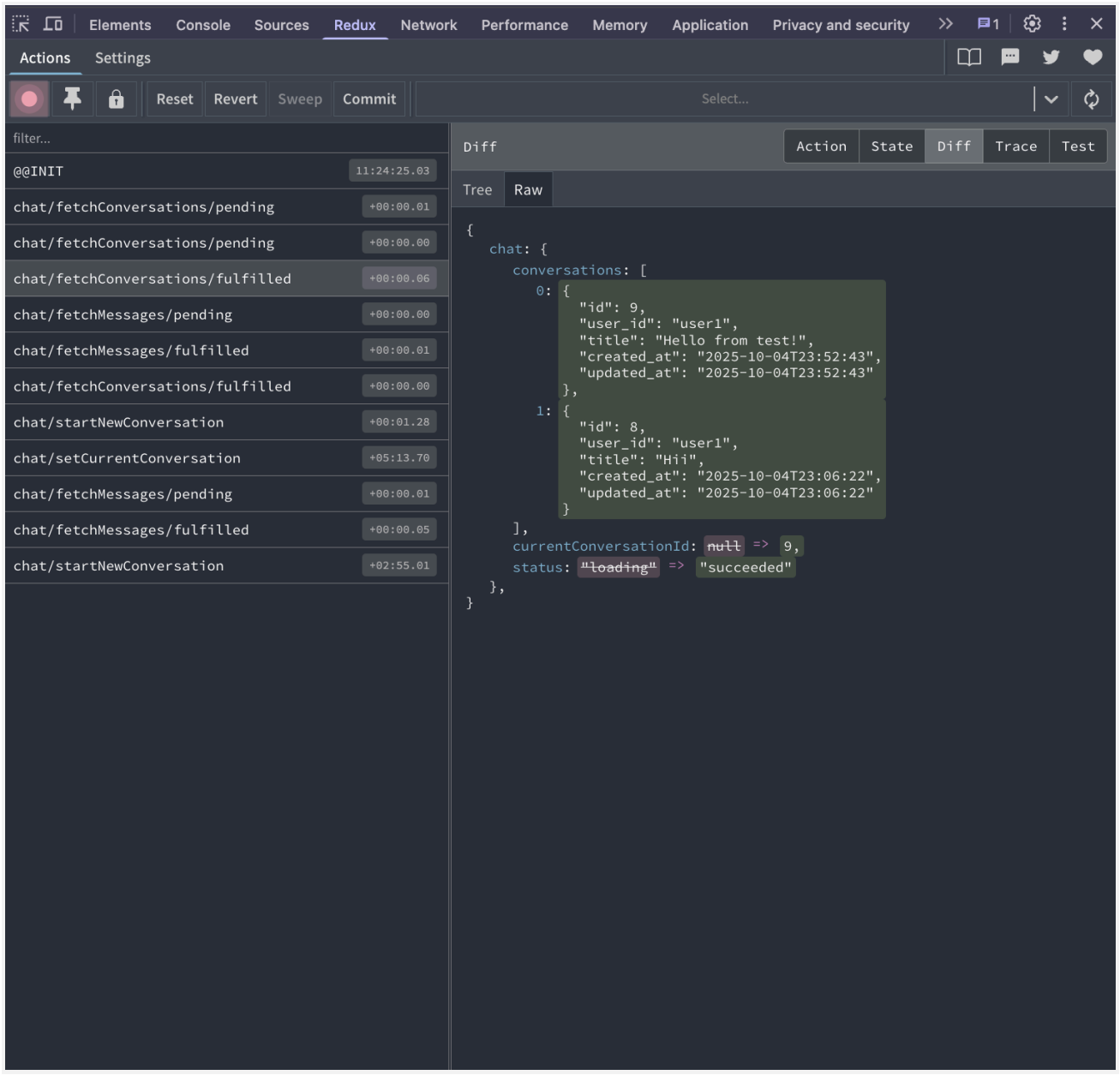


Frontend — Redux Slice

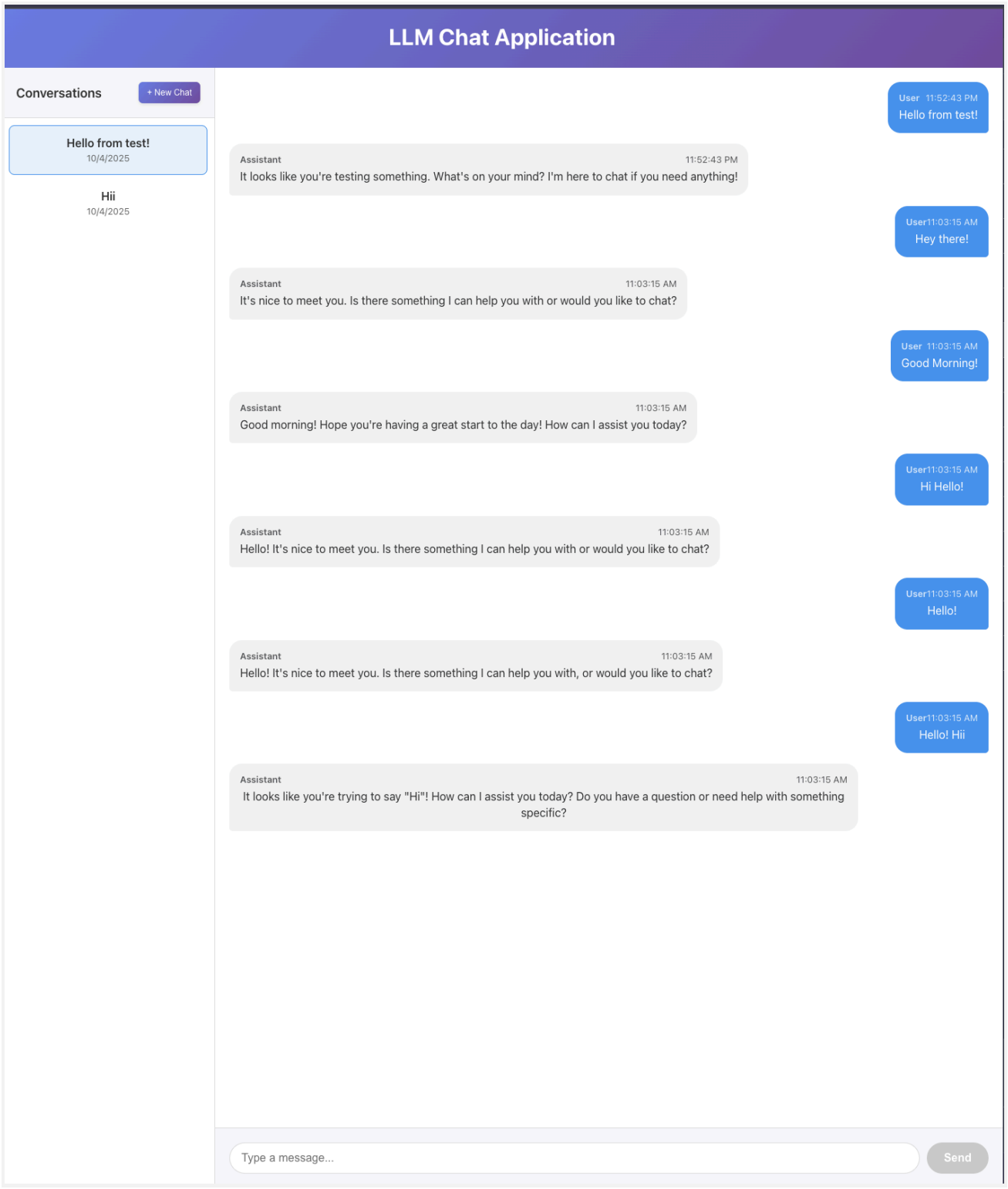
○ fetchConversations, fetchMessages, sendMessage thunks using Axios.

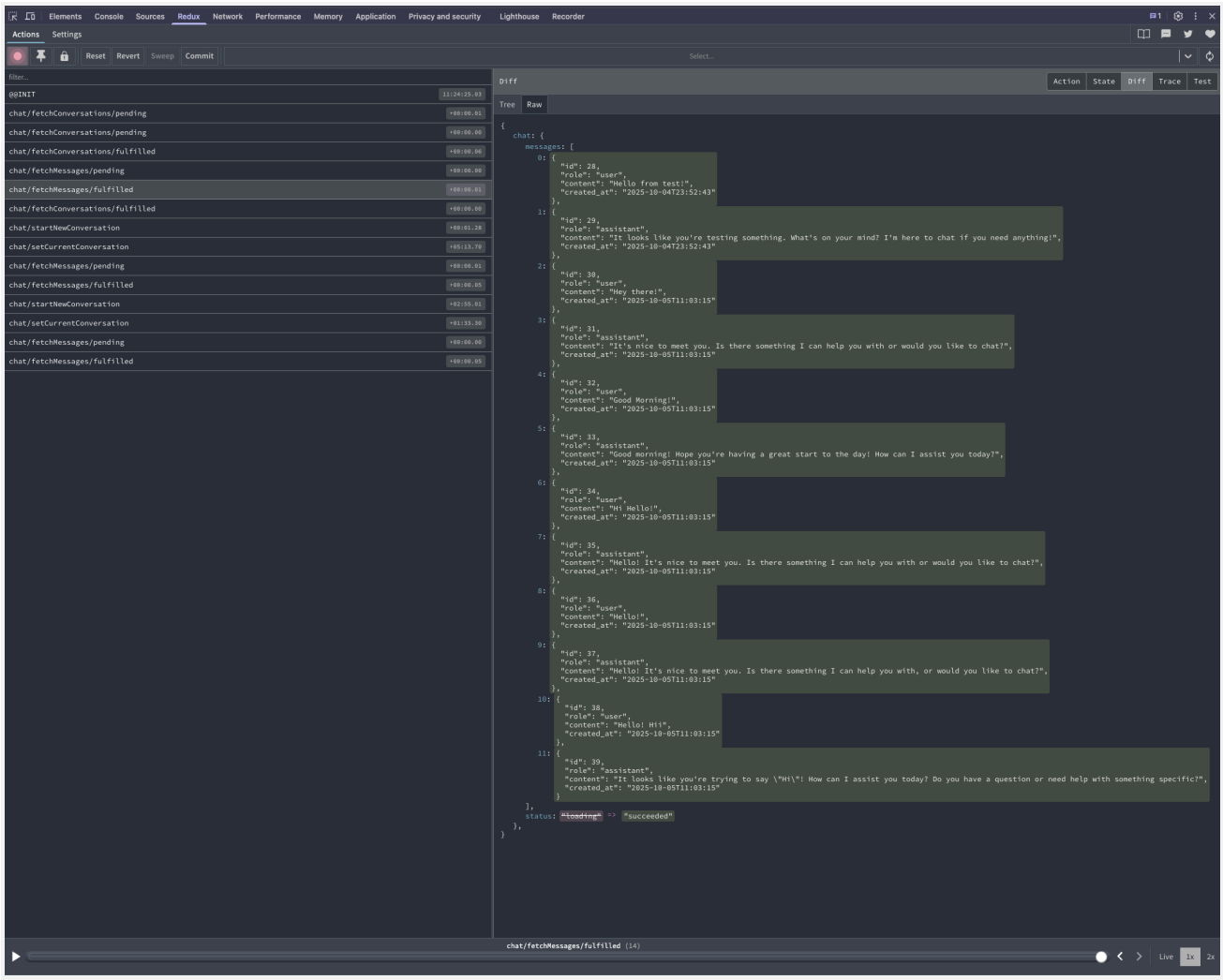
Fetch Conversations:



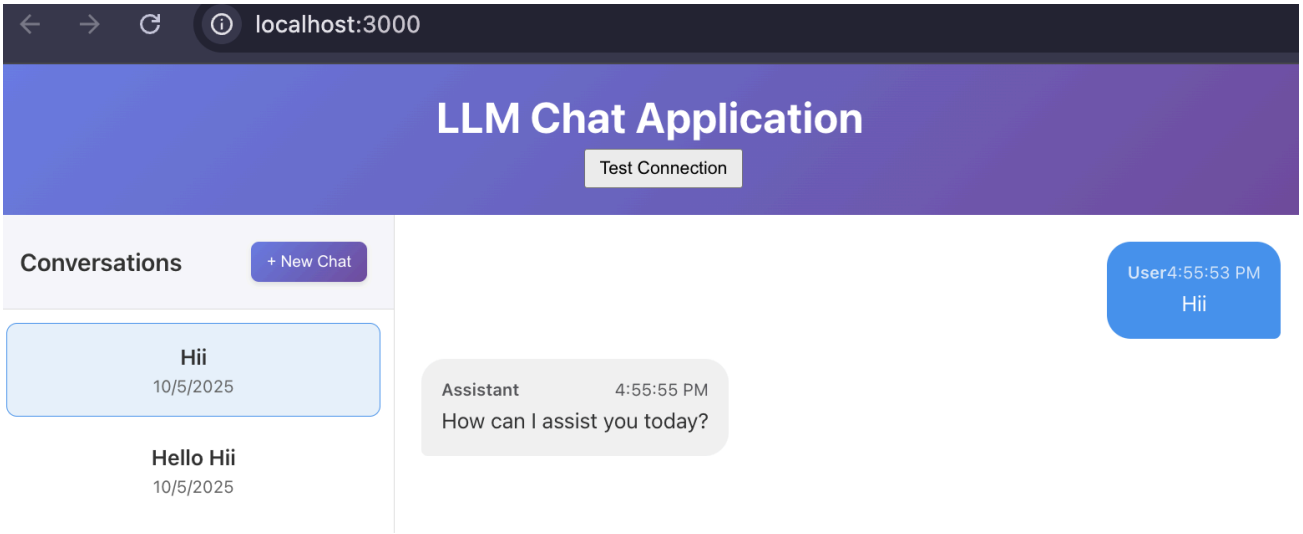

Elements
Console
Sources
Redux
Network
Performance
Memory
Application
Privacy and security
Actions
Settings
Reset
Revert
Sweep
Commit
Select...
filter...
@@INIT 11:24:25.03
chat/fetchConversations/pending +00:00.01
chat/fetchConversations/pending +00:00.00
chat/fetchConversations/fulfilled +00:00.06
chat/fetchMessages/pending +00:00.00
chat/fetchMessages/fulfilled +00:00.01
chat/fetchConversations/fulfilled +00:00.00
chat/startNewConversation +00:01.28
chat/setCurrentConversation +05:13.70
chat/fetchMessages/pending +00:00.01
chat/fetchMessages/fulfilled +00:00.05
chat/startNewConversation +02:55.01
Diff
Action
State
Diff
Trace
Test
Tree
Raw
{
chat: {
conversations: [
0: {
"id": 9,
"user_id": "user1",
"title": "Hello from test!",
"created_at": "2025-10-04T23:52:43",
"updated_at": "2025-10-04T23:52:43"
},
1: {
"id": 8,
"user_id": "user1",
"title": "Hii",
"created_at": "2025-10-04T23:06:22",
"updated_at": "2025-10-04T23:06:22"
}
],
currentConversationId: null => 9,
status: "loading" => "succeeded"
},
}

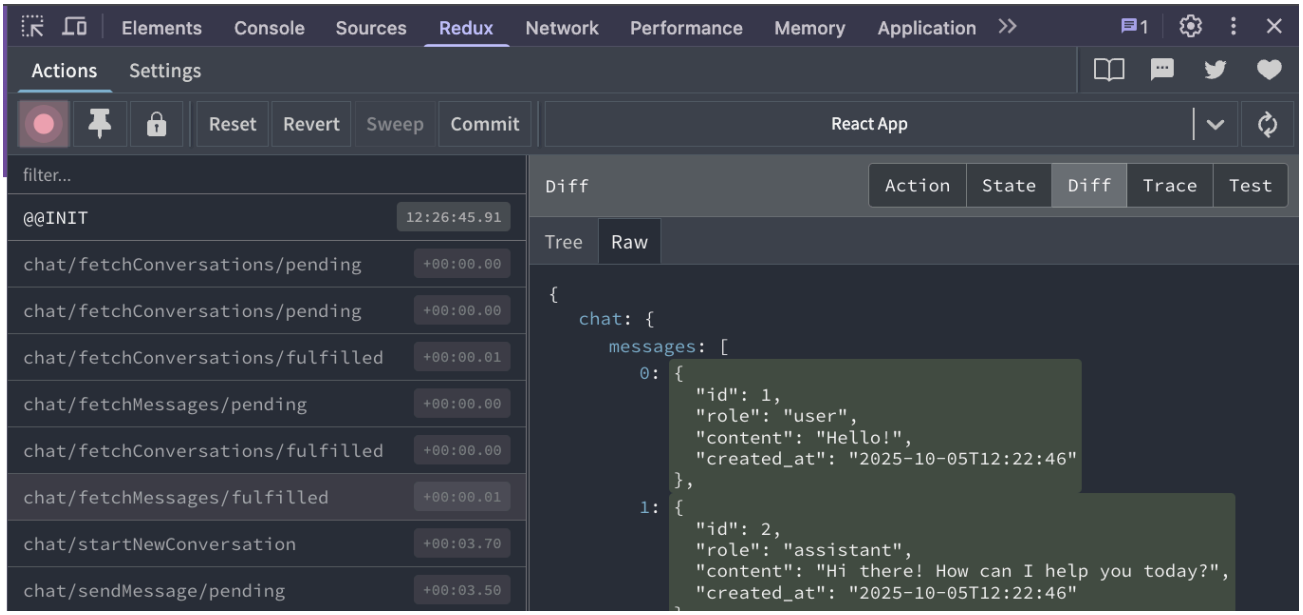
Fetch Messages:



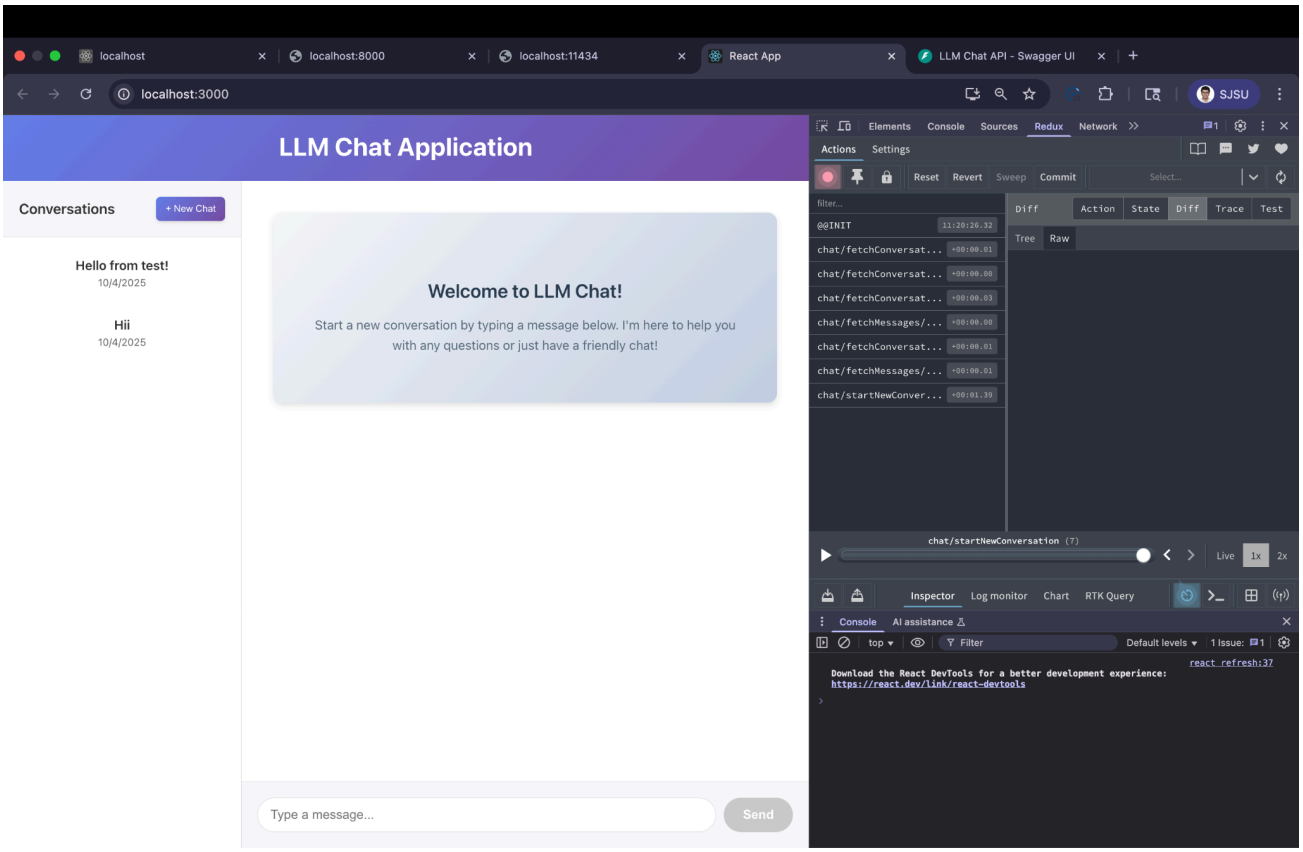

Elements
Console
Sources
Redux
Network
Performance
Memory
Application
Privacy and security
Lighthouse
Recorder
Actions
Settings
Reset
Revert
Sweep
Commit
Select...
filter...
@@INIT 11:24:25.03
chat/fetchConversations/pending +00:00.01
chat/fetchConversations/pending +00:00.00
chat/fetchConversations/fulfilled +00:00.06
chat/fetchMessages/pending +00:00.00
chat/fetchMessages/fulfilled +00:00.01
chat/fetchConversations/fulfilled +00:00.00
chat/startNewConversation +00:01.28
chat/setCurrentConversation +05:13.70
chat/fetchMessages/pending +00:00.01
chat/fetchMessages/fulfilled +00:00.05
chat/startNewConversation +02:55.01
chat/setCurrentConversation +01:33.30
chat/fetchMessages/pending +00:00.00
chat/fetchMessages/fulfilled +00:00.05
Diff
Action
State
Diff
Trace
Test
Tree
Raw
{
chat: {
messages: [
0: {
"id": 28,
"role": "user",
"content": "Hello from test!",
"created_at": "2025-10-04T23:52:43"
},
1: {
"id": 29,
"role": "assistant",
"content": "It looks like you're testing something. What's on your mind? I'm here to chat if you need anything!",
"created_at": "2025-10-04T23:52:43"
},
2: {
"id": 30,
"role": "user",
"content": "Hey there!",
"created_at": "2025-10-05T11:03:15"
},
3: {
"id": 31,
"role": "assistant",
"content": "It's nice to meet you. Is there something I can help you with or would you like to chat?",
"created_at": "2025-10-05T11:03:15"
},
4: {
"id": 32,
"role": "user",
"content": "Good Morning!",
"created_at": "2025-10-05T11:03:15"
},
5: {
"id": 33,
"role": "assistant",
"content": "Good morning! Hope you're having a great start to the day! How can I assist you today?",
"created_at": "2025-10-05T11:03:15"
},
6: {
"id": 34,
"role": "user",
"content": "Hi Hello!",
"created_at": "2025-10-05T11:03:15"
},
7: {
"id": 35,
"role": "assistant",
"content": "Hello! It's nice to meet you. Is there something I can help you with or would you like to chat?",
"created_at": "2025-10-05T11:03:15"
},
8: {
"id": 36,
"role": "user",
"content": "Hello!",
"created_at": "2025-10-05T11:03:15"
},
9: {
"id": 37,
"role": "assistant",
"content": "Hello! It's nice to meet you. Is there something I can help you with, or would you like to chat?",
"created_at": "2025-10-05T11:03:15"
},
10: {
"id": 38,
"role": "user",
"content": "Hello! Hii",
"created_at": "2025-10-05T11:03:15"
},
11: {
"id": 39,
"role": "assistant",
"content": "It looks like you're trying to say \"Hi\"! How can I assist you today? Do you have a question or need help with something specific?",
"created_at": "2025-10-05T11:03:15"
}
],
status: "loading" => "succeeded"
},
}
chat/fetchMessages/fulfilled (14)
Live
1x
2x

sendMessage:





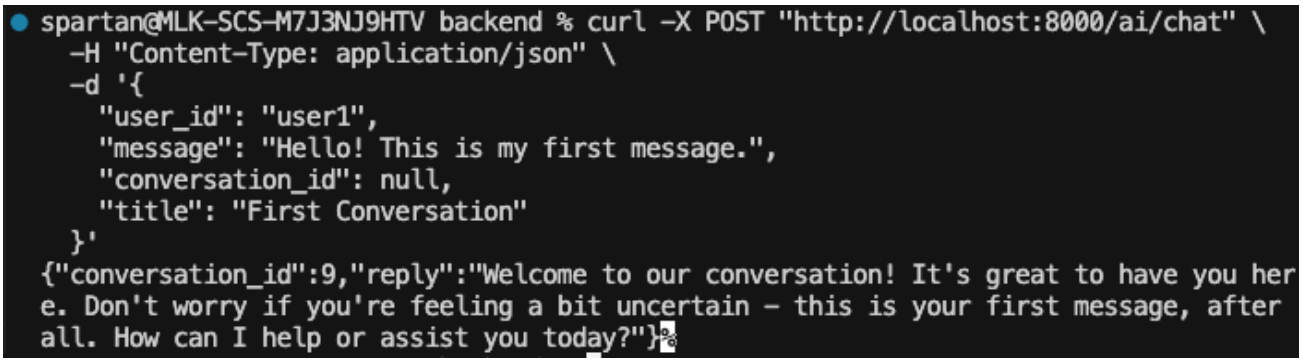
LLM Chat UI:



Testing

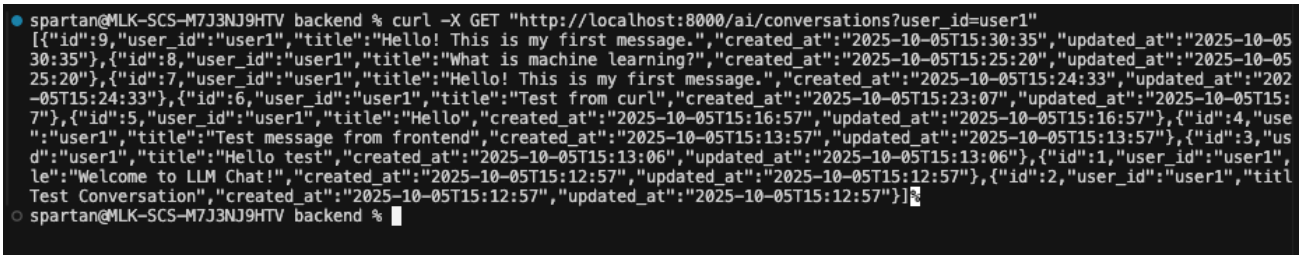
cURL examples:

1)Start New Conversation:



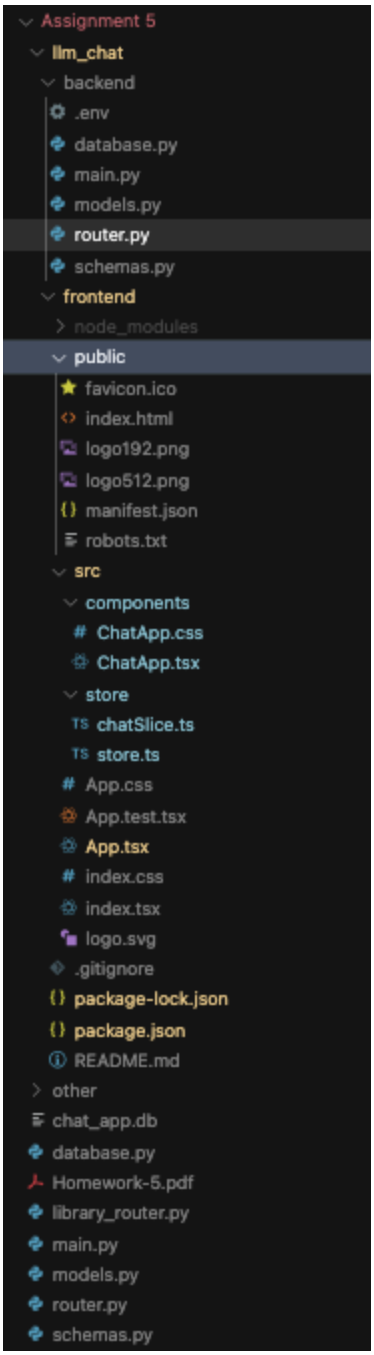
2)Continue Existing Conversation:

3)List All Conversations:



4)fetch messages:

CODE:


Assignment 5
llm_chat
backend
.env
database.py
main.py
models.py
router.py
schemas.py
frontend
node_modules
public
favicon.ico
index.html
logo192.png
logo512.png
manifest.json
robots.txt
src
components
ChatApp.css
ChatApp.tsx
store
chatSlice.ts
store.ts
App.css
App.test.tsx
App.tsx
index.css
index.tsx
logo.svg
.gitignore
package-lock.json
package.json
README.md
other
chat_app.db
database.py
Homework-5.pdf
library_router.py
main.py
models.py
router.py
schemas.py

router.py .../backend | ChatApp.tsx U | tsconfig.json | library_router.py | database.py | main.py .../As

Assignment > Assignment 5 > llm_chat > backend > router.py > chat

```python
from fastapi import APIRouter, Depends, HTTPException
from sqlalchemy.orm import Session
from typing import List
from datetime import datetime
import models
import schemas
from database import get_db
import httpx

router = APIRouter(prefix="/ai", tags=["ai"])

@router.get("/test")
def test_endpoint():
    try:
        from database import get_db
        from sqlalchemy.orm import Session
        db = next(get_db())

        conversations = db.query(models.Conversation).all()

        return {
            "status": "success",
            "message": "Database connection working",
            "conversations_count": len(conversations)
        }
    except Exception as e:
        return {
            "status": "error",
            "message": str(e),
            "error_type": type(e).__name__
        }

async def call_ollama(message: str, model: str = "llama3.1:latest") -> str:
    async with httpx.AsyncClient(timeout=httpx.Timeout(120.0)) as client:
        try:
            response = await client.post(
                "http://localhost:11434/api/generate",
                json={
                    "model": model,
                    "prompt": message,
                    "stream": False
                }
            )
            response.raise_for_status()
            data = response.json()
            return data.get("response", "")
        except httpx.RequestError as e:
            raise HTTPException(status_code=503, detail=f"Ollama service unavailable: {str(e)}")
        except httpx.HTTPStatusError as e:
            raise HTTPException(status_code=e.response.status_code, detail=f"Ollama API error: {str(e)}")

@router.post("/chat", response_model=schemas.ChatOut)
async def chat(input: schemas.ChatIn, db: Session = Depends(get_db)):
    try:
        if input.conversation_id:
            conv = db.query(models.Conversation).filter_by(id=input.conversation_id).first()
            if not conv:
                raise HTTPException(status_code=404, detail="Conversation not found")
        else:
            now = datetime.now()
            conv = models.Conversation(
                user_id=input.user_id,
                title=input.title or input.message[:40] + "..." if len(input.message) > 40 else input.message,
                created_at=now,
                updated_at=now
            )
            db.add(conv)
            db.commit()
            db.refresh(conv)

        now = datetime.now()
        user_msg = models.Message(
            conversation_id=conv.id,
            role=models.RoleEnum.user,
            content=input.message,
            created_at=now
        )
        db.add(user_msg)
        db.commit()
```

router.py .../backend | ChatApp.tsx U | tsconfig.json | library_router.py | database.py | main.py .../Assignment 5

Assignment > Assignment 5 > llm_chat > backend > router.py > chat

```python
async def chat(input: schemas.ChatIn, db: Session = Depends(get_db)):
        )
        db.add(user_msg)
        db.commit()

        try:
            reply = await call_ollama(input.message)
        except HTTPException:
            raise
        except Exception as e:
            raise HTTPException(status_code=500, detail=f"Error calling Ollama: {str(e)}")

        now = datetime.now()
        assistant_msg = models.Message(
            conversation_id=conv.id,
            role=models.RoleEnum.assistant,
            content=reply,
            created_at=now
        )
        db.add(assistant_msg)
        db.commit()

        return schemas.ChatOut(conversation_id=conv.id, reply=reply)
    except Exception as e:
        print(f"Error in chat endpoint: {e}")
        import traceback
        traceback.print_exc()
        raise HTTPException(status_code=500, detail=f"Internal server error: {str(e)}")

@router.get("/conversations", response_model=List[schemas.ConversationOut])
def get_conversations(user_id: str, db: Session = Depends(get_db)):
    conversations = db.query(models.Conversation).filter_by(user_id=user_id).all()

    valid_conversations = []
    for conv in conversations:
        try:
            if conv.created_at is None or conv.updated_at is None:
                continue
            if str(conv.created_at) == 'None' or str(conv.updated_at) == 'None':
                continue
            if str(conv.created_at) == '' or str(conv.updated_at) == '':
                continue
            if not isinstance(conv.created_at, datetime) or not isinstance(conv.updated_at, datetime):
                continue
            valid_conversations.append(conv)
        except Exception as e:
            continue

    valid_conversations.sort(key=lambda x: x.updated_at, reverse=True)

    return valid_conversations

@router.get("/messages/{conversation_id}", response_model=schemas.MessagesOut)
def get_messages(conversation_id: int, db: Session = Depends(get_db)):
    """Get all messages in a conversation"""
    conv = db.query(models.Conversation).filter_by(id=conversation_id).first()
    if not conv:
        raise HTTPException(status_code=404, detail="Conversation not found")

    messages = db.query(models.Message).filter_by(conversation_id=conversation_id).order_by(models.Message.created_at.asc()).all()

    valid_messages = []
    for m in messages:
        try:
            if m.created_at is None:
                continue
            if str(m.created_at) == 'None':
                continue
            if str(m.created_at) == '':
                continue
            valid_messages.append(m)
        except Exception as e:
            continue

    return schemas.MessagesOut(
        conversation_id=conversation_id,
        messages=[
```

```python
    valid_messages = []
    for m in messages:
        try:
            if m.created_at is None:
                continue
            if str(m.created_at) == 'None':
                continue
            if str(m.created_at) == '':
                continue
            valid_messages.append(m)
        except Exception as e:
            continue

    return schemas.MessagesOut(
        conversation_id=conversation_id,
        messages=[
            schemas.ChatMessageOut(
                id=m.id,
                role=m.role.value,
                content=m.content,
                created_at=m.created_at
            ) for m in valid_messages
        ]
    )

```

models.py .../Assignment 5 | schemas.py .../backend | router.py .../Assig

Assignment > Assignment 5 > llm_chat > backend > schemas.py > ChatIn

```python
from pydantic import BaseModel, field_validator
from datetime import datetime
from typing import Optional, List

class ChatIn(BaseModel):
    user_id: str
    message: str
    conversation_id: Optional[int] = None
    title: Optional[str] = None

class ChatOut(BaseModel):
    conversation_id: int
    reply: str

class ChatMessageOut(BaseModel):
    id: int
    role: str
    content: str
    created_at: datetime

    @field_validator('created_at', mode='before')
    @classmethod
    def validate_datetime(cls, v):
        if v is None or v == '' or v == 'None':
            return datetime.now()
        return v

class MessagesOut(BaseModel):
    conversation_id: int
    messages: List[ChatMessageOut]

class ConversationOut(BaseModel):
    id: int
    user_id: str
    title: Optional[str]
    created_at: datetime
    updated_at: datetime

    @field_validator('created_at', 'updated_at', mode='before')
    @classmethod
    def validate_datetime(cls, v):
        if v is None or v == '' or v == 'None':
            return datetime.now()
        return v

    class Config:
        from_attributes = True

```

database.py .../backend 1 | router.py .../Assignment 5 | schemas.py .../Assignment 5 | TS

Assignment > Assignment 5 > llm_chat > backend > database.py > get_db

```python
from sqlalchemy import create_engine
from sqlalchemy.orm import sessionmaker
import os
from dotenv import load_dotenv

load_dotenv()

DB_HOST = os.getenv("DB_HOST", "localhost")
DB_PORT = os.getenv("DB_PORT", "3306")
DB_USER = os.getenv("DB_USER", "root")
DB_PASSWORD = os.getenv("DB_PASSWORD", "pass1234")
DB_NAME = os.getenv("DB_NAME", "book_db")

MYSQL_URL = f"mysql+pymysql://{DB_USER}:{DB_PASSWORD}@{DB_HOST}:{DB_PORT}/{DB_NAME}"

engine = create_engine(SQLALCHEMY_DATABASE_URL)
SessionLocal = sessionmaker(autocommit=False, autoflush=False, bind=engine)

def get_db():
    db = SessionLocal()
    try:
        yield db
    finally:
        db.close()

```

models.py .../backend | router.py .../backend | ChatApp.tsx U | library_router.py | database.py

Assignment > Assignment 5 > llm_chat > backend > models.py > ...

```python
from sqlalchemy import Column, Integer, String, ForeignKey, Enum, Text, DateTime, func
from sqlalchemy.orm import relationship, declarative_base
import enum

Base = declarative_base()

class RoleEnum(enum.Enum):
    user = "user"
    assistant = "assistant"
    system = "system"

class Conversation(Base):
    __tablename__ = "conversations"

    id = Column(Integer, primary_key=True, index=True)
    user_id = Column(String(64), nullable=False)
    title = Column(String(128), nullable=True)
    created_at = Column(DateTime(timezone=True), server_default=func.now())
    updated_at = Column(DateTime(timezone=True), server_default=func.now(), onupdate=func.now())

    messages = relationship("Message", back_populates="conversation", cascade="all, delete-orphan")

class Message(Base):
    __tablename__ = "messages"

    id = Column(Integer, primary_key=True, index=True)
    conversation_id = Column(Integer, ForeignKey("conversations.id"), nullable=False)
    role = Column(Enum(RoleEnum), nullable=False)
    content = Column(Text, nullable=False)
    created_at = Column(DateTime(timezone=True), server_default=func.now())

    conversation = relationship("Conversation", back_populates="messages")
```

main.py .../backend | models.py .../backend | router.py .../backend

Assignment > Assignment 5 > llm_chat > backend > main.py > health_check

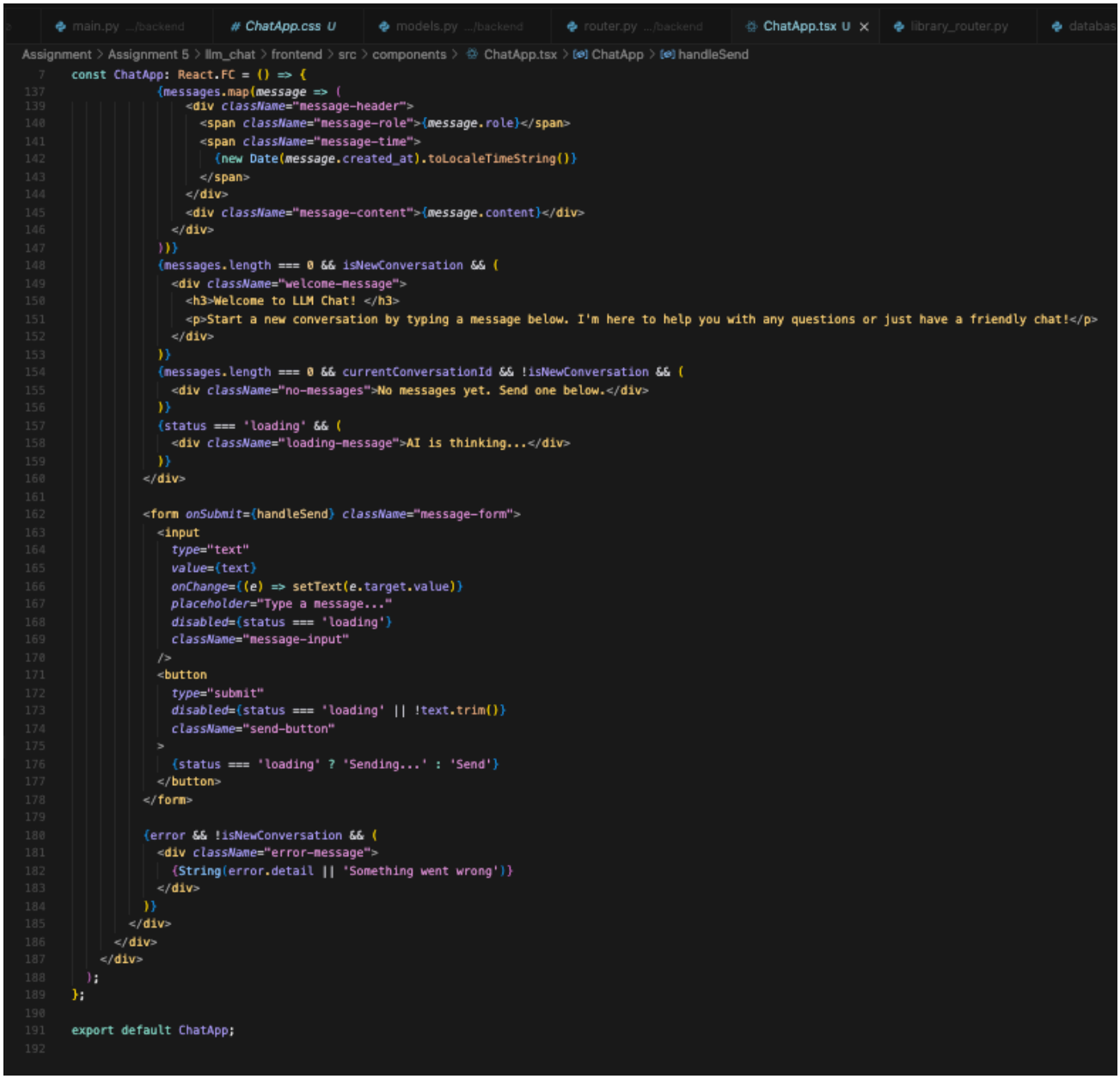
```python
from fastapi import FastAPI
from fastapi.middleware.cors import CORSMiddleware
from database import engine
from models import Base
import router

app = FastAPI(
    title="LLM Chat API",
    version="1.0.0"
)

app.add_middleware(
    CORSMiddleware,
    allow_origins=["*"],
    allow_credentials=True,
    allow_methods=["*"],
    allow_headers=["*"],
)

app.include_router(router.router)

@app.on_event("startup")
def on_startup():
    Base.metadata.create_all(bind=engine)
    print("Database tables created successfully!")
    print("LLM Chat API is ready!")

@app.get("/")
def read_root():
    return {"message": "LLM Chat API is running!", "docs": "/docs"}

@app.get("/health")
def health_check():
    return {"status": "healthy", "service": "LLM Chat API"}

if __name__ == "__main__":
    import uvicorn
    uvicorn.run(app, host="0.0.0.0", port=8000)
```


main.py .../backend
ChatApp.css U
models.py .../backend
router.py .../backend
ChatApp.tsx U
library_router.py
database.py
Assignment > Assignment 5 > llm_chat > frontend > src > components > ChatApp.tsx > ChatApp > handleSend
import React, { useEffect, useState } from 'react';
import { useDispatch, useSelector } from 'react-redux';
import { fetchConversations, fetchMessages, sendMessage, setCurrentConversation, startNewConversation } from '../store/chatSlice';
import { RootState, AppDispatch } from '../store/store';
import './ChatApp.css';
const ChatApp: React.FC = () => {
const dispatch = useDispatch<AppDispatch>();
const { conversations, messages, currentConversationId, isNewConversation, userId, status, error } = useSelector((state: RootState) => state.chat);
const [text, setText] = useState('');
useEffect(() => {
dispatch(fetchConversations(userId));
}, [dispatch, userId]);
useEffect(() => {
if (currentConversationId && !isNewConversation) {
dispatch(fetchMessages(currentConversationId));
}
}, [dispatch, currentConversationId, isNewConversation]);
useEffect(() => {
if (currentConversationId) {
dispatch(fetchConversations(userId));
}
}, [dispatch, currentConversationId, userId]);
const handleSend = async (e: React.FormEvent) => {
e.preventDefault();
if (!text.trim()) return;
console.log('ChatApp: Starting send message process...');
console.log('ChatApp: Current state:', {
userId,
text,
currentConversationId,
isNewConversation
});
const messageText = text;
setText('');
try {
const result = await dispatch(sendMessage({
userId,
message: messageText,
conversationId: currentConversationId || undefined,
title: messageText.slice(0, 40)
}));
console.log('ChatApp: Send message result:', result);
if (sendMessage.fulfilled.match(result)) {
console.log('ChatApp: Message sent successfully, fetching messages...');
if (result.payload.conversation_id) {
await dispatch(fetchMessages(result.payload.conversation_id));
}
await dispatch(fetchConversations(userId));
} else if (sendMessage.rejected.match(result)) {
console.log('ChatApp: Message send rejected:', result.payload);
} else {
console.log('ChatApp: Message send in unknown state:', result);
}
} catch (error) {
console.error('ChatApp: Error in handleSend:', error);
}
};
const handleConversationSelect = (conversationId: number) => {
dispatch(setCurrentConversation(conversationId));
};
const handleNewConversation = () => {
dispatch(startNewConversation());
setText('');
};

main.py .../backend | # ChatApp.css U | models.py .../backend | router.py .../backend | ChatApp.tsx U

Assignment > Assignment 5 > llm_chat > frontend > src > components > ChatApp.tsx > ChatApp > handleSend

```
  const ChatApp: React.FC = () => {
    const handleNewConversation = () => {
      dispatch(startNewConversation());
      setText('');
    };

    const testConnection = async () => {
      try {
        console.log('Testing backend connection...');
        const response = await fetch('http://localhost:8000/health');
        const data = await response.json();
        console.log('Backend health check:', data);
        alert(`Backend is running! Status: ${data.status}`);
      } catch (error) {
        console.error('Backend connection failed:', error);
        const errorMessage = error instanceof Error ? error.message : 'Unknown error';
        alert(`Backend connection failed: ${errorMessage}`);
      }
    };

    return (
      <div className="chat-app">
        <div className="chat-header">
          <h1>LLM Chat Application</h1>
          <button onClick={testConnection} style={{marginLeft: '20px', padding: '5px 10px'}}>
            Test Connection
          </button>
        </div>

        <div className="chat-container">
          {}
          <div className="conversations-sidebar">
            <div className="conversations-header">
              <h3>Conversations</h3>
              <button
                onClick={handleNewConversation}
                className="new-conversation-btn"
                title="Start a new conversation"
              >
                + New Chat
              </button>
            </div>
            <div className="conversations-list">
              {conversations.map(conv => (
                <div
                  key={conv.id}
                  className={`conversation-item ${currentConversationId === conv.id ? 'active' : ''}`}
                  onClick={() => handleConversationSelect(conv.id)}
                >
                  <div className="conversation-title">{conv.title || 'New Chat'}</div>
                  <div className="conversation-date">
                    {new Date(conv.created_at).toLocaleDateString()}
                  </div>
                </div>
              ))}
              {conversations.length === 0 && (
                <div className="no-conversations">No conversations yet</div>
              )}
            </div>
          </div>


          <div className="chat-main">
            <div className="messages-container">
              {messages.map(message => (
                <div key={message.id} className={`message ${message.role}`}>
                  <div className="message-header">
                    <span className="message-role">{message.role}</span>
                    <span className="message-time">
                      {new Date(message.created_at).toLocaleTimeString()}
                    </span>
                  </div>
                  <div className="message-content">{message.content}</div>
                </div>
              ))}
              {messages.length === 0 && isNewConversation && (
```


main.py .../backend
ChatApp.css U
models.py .../backend
router.py .../backend
ChatApp.tsx U
library_router.py
Assignment > Assignment 5 > llm_chat > frontend > src > components > ChatApp.tsx > ChatApp > handleSend
const ChatApp: React.FC = () => {
{messages.map(message => (

{message.role}

{new Date(message.created_at).toLocaleTimeString()}

{message.content}

))}
{messages.length === 0 && isNewConversation && (

<h3>Welcome to LLM Chat! </h3>
Start a new conversation by typing a message below. I'm here to help you with any questions or just have a friendly chat!

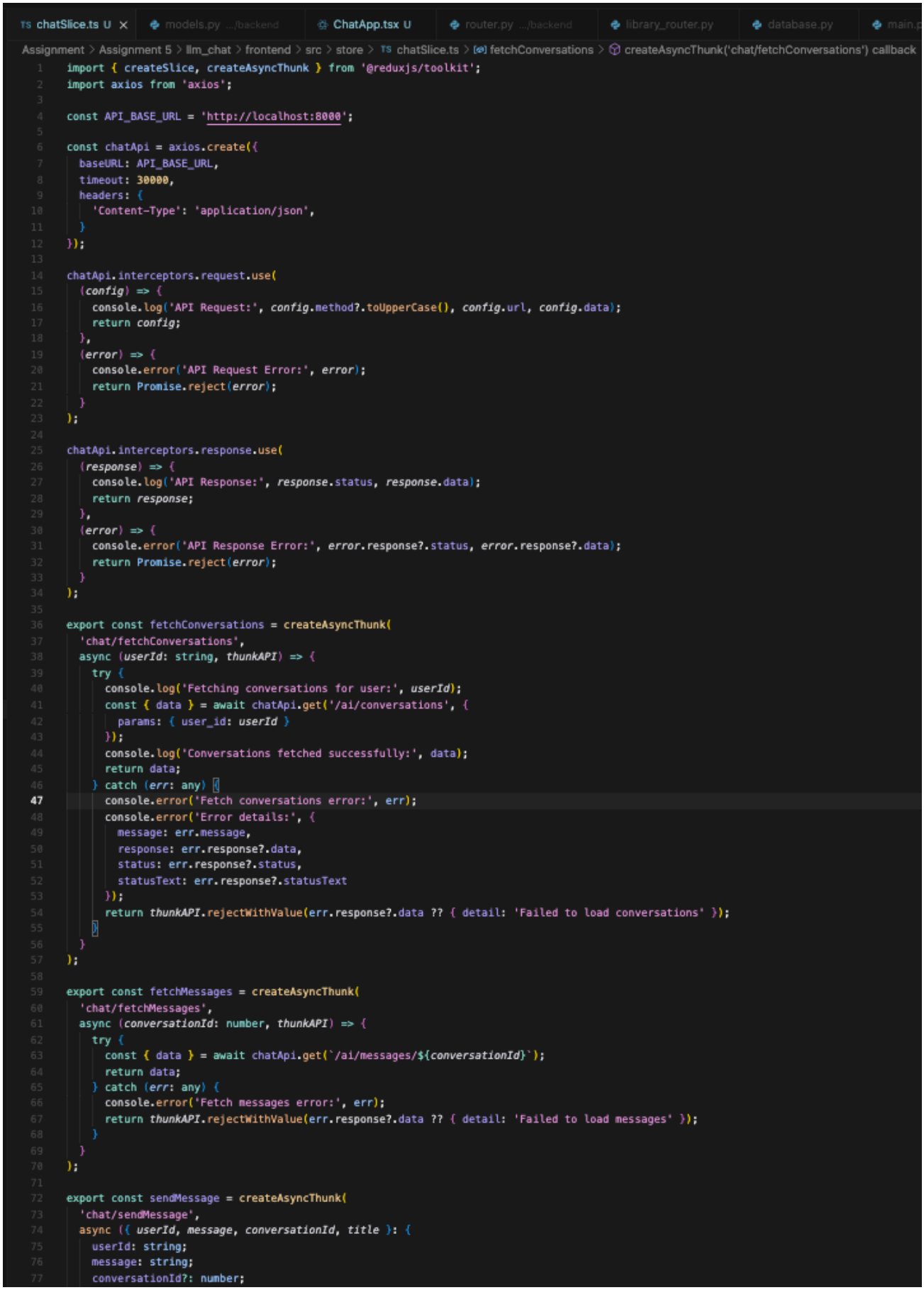
)}
{messages.length === 0 && currentConversationId && !isNewConversation && (
No messages yet. Send one below.
)}
{status === 'loading' && (
AI is thinking...
)}

<form onSubmit={handleSend} className="message-form">
<input
type="text"
value={text}
onChange={(e) => setText(e.target.value)}
placeholder="Type a message..."
disabled={status === 'loading'}
className="message-input"
/>
<button
type="submit"
disabled={status === 'loading' || !text.trim()}
className="send-button"
>
{status === 'loading' ? 'Sending...' : 'Send'}
</button>
</form>
{error && !isNewConversation && (

{String(error.detail || 'Something went wrong')}

)}

);
};
export default ChatApp;


chatSlice.ts U
models.py .../backend
ChatApp.tsx U
router.py .../backend
library_router.py
database.py
Assignment > Assignment 5 > llm_chat > frontend > src > store > chatSlice.ts > fetchConversations > createAsyncThunk('chat/fetchConversations') callback
import { createSlice, createAsyncThunk } from '@reduxjs/toolkit';
import axios from 'axios';
const API_BASE_URL = 'http://localhost:8000';
const chatApi = axios.create({
baseURL: API_BASE_URL,
timeout: 30000,
headers: {
'Content-Type': 'application/json',
}
});
chatApi.interceptors.request.use(
(config) => {
console.log('API Request:', config.method?.toUpperCase(), config.url, config.data);
return config;
},
(error) => {
console.error('API Request Error:', error);
return Promise.reject(error);
}
);
chatApi.interceptors.response.use(
(response) => {
console.log('API Response:', response.status, response.data);
return response;
},
(error) => {
console.error('API Response Error:', error.response?.status, error.response?.data);
return Promise.reject(error);
}
);
export const fetchConversations = createAsyncThunk(
'chat/fetchConversations',
async (userId: string, thunkAPI) => {
try {
console.log('Fetching conversations for user:', userId);
const { data } = await chatApi.get('/ai/conversations', {
params: { user_id: userId }
});
console.log('Conversations fetched successfully:', data);
return data;
} catch (err: any) {
console.error('Fetch conversations error:', err);
console.error('Error details:', {
message: err.message,
response: err.response?.data,
status: err.response?.status,
statusText: err.response?.statusText
});
return thunkAPI.rejectWithValue(err.response?.data ?? { detail: 'Failed to load conversations' });
}
}
);
export const fetchMessages = createAsyncThunk(
'chat/fetchMessages',
async (conversationId: number, thunkAPI) => {
try {
const { data } = await chatApi.get(`/ai/messages/${conversationId}`);
return data;
} catch (err: any) {
console.error('Fetch messages error:', err);
return thunkAPI.rejectWithValue(err.response?.data ?? { detail: 'Failed to load messages' });
}
}
);
export const sendMessage = createAsyncThunk(
'chat/sendMessage',
async ({ userId, message, conversationId, title }: {
userId: string;
message: string;
conversationId?: number;

```
export const sendMessage = createAsyncThunk(
  async ({ userId, message, conversationId, title }: {
    message: string;
    conversationId?: number;
    title?: string;
  }, thunkAPI) => {
    console.log('Redux: Sending message to API:', { userId, message, conversationId, title });

    const payload = {
      user_id: userId,
      message,
      conversation_id: conversationId ?? null,
      title: title ?? message.slice(0, 40),
    };

    console.log('Redux: Payload prepared:', payload);

    try {
      const response = await chatApi.post('/ai/chat', payload);
      console.log('Redux: API response received:', response.data);

      console.log('Redux: Returning data:', {
        conversation_id: response.data.conversation_id,
        reply: response.data.reply
      });

      return {
        conversation_id: response.data.conversation_id,
        reply: response.data.reply,
        success: true
      };
    } catch (error) {
      console.error('Redux: sendMessage error caught:', error);
      return {
        conversation_id: conversationId || 0,
        reply: 'Error: Failed to send message',
        success: false,
        error: error instanceof Error ? error.message : 'Unknown error'
      };
    }
  }
);

const chatSlice = createSlice({
  name: 'chat',
  initialState: {
    userId: 'user1',
    conversations: [] as any[],
    messages: [] as any[],
    currentConversationId: null as number | null,
    isNewConversation: false,
    status: 'idle' as 'idle' | 'loading' | 'succeeded' | 'failed',
    error: null as any,
  },
  reducers: {
    setCurrentConversation: (state, action) => {
      state.currentConversationId = action.payload;
      state.isNewConversation = action.payload === null;
    },
    startNewConversation: (state) => {
      state.currentConversationId = null;
      state.isNewConversation = true;
      state.messages = [];
    },
    clearError: (state) => {
      state.error = null;
    },
  },
  extraReducers: (builder) => {
    builder
      .addCase(fetchConversations.pending, (state) => {
        state.status = 'loading';
        state.error = null;
      })
      .addCase(fetchConversations.fulfilled, (state, action) => {
        state.status = 'succeeded';
        state.conversations = action.payload;
```


chatSlice.ts U
models.py .../backend
ChatApp.tsx U
router.py .../backend
library_router.py
database.py
Assignment > Assignment 5 > llm_chat > frontend > src > store > chatSlice.ts > fetchConversations > createAsyncThunk('chat/fetchConversations') callback
const chatSlice = createSlice({
extraReducers: (builder) => {
builder
.addCase(fetchConversations.pending, (state) => {
state.status = 'loading';
state.error = null;
})
.addCase(fetchConversations.fulfilled, (state, action) => {
state.status = 'succeeded';
state.conversations = action.payload;
if (!state.isNewConversation && state.currentConversationId === null && action.payload.length > 0) {
state.currentConversationId = action.payload[0].id;
}
})
.addCase(fetchConversations.rejected, (state, action) => {
state.status = 'failed';
state.error = action.payload || action.error;
})
.addCase(fetchMessages.pending, (state) => {
state.status = 'loading';
state.error = null;
})
.addCase(fetchMessages.fulfilled, (state, action) => {
state.status = 'succeeded';
state.messages = action.payload.messages || [];
state.currentConversationId = action.payload.conversation_id ?? state.currentConversationId;
state.isNewConversation = false;
})
.addCase(fetchMessages.rejected, (state, action) => {
state.status = 'failed';
state.error = action.payload || action.error;
if (state.isNewConversation) {
state.status = 'succeeded';
state.error = null;
}
})
.addCase(sendMessage.pending, (state) => {
console.log('Redux: sendMessage.pending');
state.status = 'loading';
state.error = null;
})
.addCase(sendMessage.fulfilled, (state, action) => {
console.log('Redux: sendMessage.fulfilled with payload:', action.payload);
state.status = 'succeeded';
state.currentConversationId = action.payload.conversation_id;
state.isNewConversation = false;
if (action.payload.success) {
state.error = null;
console.log('Redux: Message sent successfully, conversation_id:', action.payload.conversation_id);
} else {
state.error = { detail: action.payload.error || 'Failed to send message' };
console.log('Redux: Message send failed:', action.payload.error);
}
})
.addCase(sendMessage.rejected, (state, action) => {
console.log('Redux: sendMessage.rejected with payload:', action.payload);
state.status = 'failed';
state.error = action.payload || action.error;
});
},
});
export const { setCurrentConversation, startNewConversation, clearError } = chatSlice.actions;
export default chatSlice.reducer;

WORKING CHAT: