

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

Name	Description
page integer (query)	Page number 1 minimum: 1
per_page integer (query)	Items per page 10 maximum: 100 minimum: 1

Cancel

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

```
{ "authors": [ { "first_name": "Harper", "last_name": "Lee", "email": "harper.lee@gmail.com", "id": 1, "created_at": "2025-10-06T05:00:20", "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 } ] }
```

Download

Response headers

```
content-length: 469
content-type: application/json
date: Mon, 06 Oct 2025 05:04:32 GMT
server: unicorn
```

Responses

Code	Description	Links
200	Successful Response	No links

GET /library/authors Get Authors

Get all authors with pagination

Parameters

Name Description

page	Page number
integer (query)	<input type="text" value="1"/> minimum: 1
per_page	Items per page
integer (query)	<input type="text" value="10"/> maximum: 100 minimum: 1

Cancel

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

```
[{"id": 1, "first_name": "George", "last_name": "Orwell", "email": "george.orwell@gmail.com", "created_at": "2025-10-06T05:00:20", "updated_at": null}, {"id": 2, "first_name": "J.K.", "last_name": "Rowling", "email": "jk.rowling@gmail.com", "created_at": "2025-10-06T05:02:35", "updated_at": null}, {"id": 3, "first_name": "Stephen", "last_name": "King", "email": "stephen.king@gmail.com", "created_at": "2025-10-06T05:03:00", "updated_at": null}], {"total": 3, "page": 1, "per_page": 10, "total_pages": 1}
```

Download

SQL Screenshot:

The screenshot shows the MySQL Workbench interface. The left sidebar displays the schema structure of the 'book_db' database, specifically the 'authors' table, which has columns: id, first_name, last_name, email, created_at, and updated_at. The main area shows the results of the SQL query: 'select * from authors;'. The results grid contains three rows of data:

	id	first_name	last_name	email	created_at	updated_at
1	1	Harper	Lee	harper.lee@gmail.com	2025-10-05 21:07:54	2025-10-05 21:07:54
2	2	George	Orwell	george.orwell@gmail.com	2025-10-05 21:07:54	2025-10-05 21:07:54
3	3	J.K.	Rowling	jk.rowling@gmail.com	2025-10-05 21:07:54	2025-10-05 21:07:54
	HULL	HULL	HULL	HULL	HULL	HULL

2. POST /library/authors - Create new author

localhost:8000/docs#/library/create_author_library_authors_post

library

POST /library/authors Create Author

Create a new author

Parameters

No parameters

Request body required

application/json

Edit Value | Schema

```
{  "first_name": "Harper",  "last_name": "Lee",  "email": "harper.lee@gmail.com"}  
}
```

Execute Clear

Responses

Curl

```
curl -X 'POST' \
  'http://localhost:8000/library/authors' \
  -H 'accept: application/json' \
  -H 'Content-Type: application/json' \
  -d '{  "first_name": "Harper",  "last_name": "Lee",  "email": "harper.lee@gmail.com"}'  
}
```

Request URL

http://localhost:8000/library/authors

Server response

Code Details

201 Response body

```
{  "first_name": "Harper",  "last_name": "Lee",  "email": "harper.lee@gmail.com",  "id": 1,  "created_at": "2025-10-06T05:00:20",  "updated_at": null}  
}
```

Download

Response headers

```
access-control-allow-credentials: true
content-length: 132
content-type: application/json
date: Mon, 06 Oct 2025 05:00:20 GMT
server: uvicorn
```

Responses

Code Description Links

201 Successful Response No links

SQL Screenshots:

The screenshot shows the MySQL Workbench interface. On the left, the 'Schemas' tree view displays the 'book_db' schema, which contains the 'authors' table. The 'Columns' section under 'authors' lists columns: id, first_name, last_name, email, created_at, and updated_at. On the right, the 'Query 1' tab contains the SQL query: 'select * from authors;'. The results grid shows one row of data:

	id	first_name	last_name	email	created_at	updated_at
1	1	Harper	Lee	harper.lee@gmail.com	2025-10-05 21:07:54	2025-10-05 21:07:54

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

GET /library/authors/{author_id} Get Author ^

Get a specific author by ID

Parameters Cancel

Name	Description
author_id * required integer (path)	1

Execute Clear

Responses

Curl

```
curl -X 'GET' \
'http://localhost:8000/library/authors/1' \
-H 'accept: application/json'
```

Request URL

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

Server response

Code Details

200 Response body

```
{  
    "first_name": "Harper",  
    "last_name": "Lee",  
    "email": "harper.lee@gmail.com",  
    "id": 1,  
    "created_at": "2025-10-06T05:00:20",  
    "updated_at": null  
}
```

Copy Download

Response headers

```
Content-length: 132  
Content-type: application/json  
Date: Mon, 06 Oct 2025 05:00:32 GMT  
Server: uvicorn
```

Responses

Code Description Links

200 Successful Response No links

SQL:

The screenshot shows the MySQL Workbench interface. On the left, the 'SCHEMAS' tree view displays the 'book_db' schema with its tables, columns, and other database objects. The 'authors' table is selected, showing its columns: id, first_name, last_name, email, created_at, and updated_at. A query window at the top contains the SQL command: 'select * from authors;'. The results grid below shows three rows of data:

	id	first_name	last_name	email	created_at	updated_at
1	1	Harper	Lee	harper.lee@gmail.com	2025-10-05 21:07:54	2025-10-05 21:07:54
2	2	George	Orwell	george.orwell@gmail.com	2025-10-05 21:07:54	2025-10-05 21:07:54
3	3	J.K.	Rowling	jk.rowling@gmail.com	2025-10-05 21:07:54	2025-10-05 21:07:54
	HULL	HULL	HULL	HULL	HULL	HULL

The interface includes various toolbars and panels for managing databases and queries.

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

PUT /library/authors/{author_id} Update Author

Update an author's information

Parameters

Name	Description
author_id <small>required</small>	integer (path)

Request body required

application/json

Edit Value | Schema

```
{  
    "first_name": "vimalanandhan",  
    "last_name": "sivanandham",  
    "email": "vimal@gmail.com"  
}
```

Execute Clear

Responses

Curl

```
curl -X 'PUT' \  
'http://localhost:8000/library/authors/1' \  
-H 'accept: application/json' \  
-H 'Content-type: application/json' \  
-d '{  
    "first_name": "vimalanandhan",  
    "last_name": "sivanandham",  
    "email": "vimal@gmail.com"  
}'
```

Request URL

http://localhost:8000/Library/authors/1

Server response

Code Details

200 Response body

```
{  
    "first_name": "vimalanandhan",  
    "last_name": "sivanandham",  
    "email": "vimal@gmail.com",  
    "id": 1,  
    "created_at": "2025-10-06T05:00:28",  
    "updated_at": "2025-10-06T05:11:58"  
}
```

Download

Response headers

```
access-control-allow-credentials: true  
content-length: 159  
content-type: application/json  
date: Mon, 06 Oct 2025 05:11:58 GMT  
server: uvicorn
```

Responses

Code Description Links

200 Successful Response No links

SQL:

The screenshot shows the MySQL Workbench interface. On the left, the 'Schemas' tree view is expanded to show the 'book_db' schema, which contains the 'authors' table. The 'Columns' section under 'authors' lists six columns: id, first_name, last_name, email, created_at, and updated_at. The 'Result Grid' tab on the right displays the results of the SQL query 'select * from authors;'. The results show three rows of data:

	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	j.k.rowling@gmail.com	2025-10-05 21:07:54	2025-10-05 21:07:54	

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

DELETE /library/authors/{author_id} Delete Author ^

Delete an author (only if they have no associated books)

Parameters

Name **Description**

author_id * required integer (path)	<input type="text" value="1"/>
---	--------------------------------

Cancel

Execute **Clear**

Responses

Curl

```
curl -X 'DELETE' \
  'http://localhost:8000/library/authors/1' \
  -H 'accept: */*'
```

Request URL

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

Server response

Code **Details**

204	Response headers <code>access-control-allow-credentials: true content-type: application/json date: Mon, 06 Oct 2025 05:25:42 GMT server: uvicorn</code>
-----	---

Responses

Code **Description** **Links**

204	Successful Response	No links
-----	---------------------	----------

SQL

BEFORE DELETE:

The screenshot shows the MySQL Workbench interface. On the left, the 'Schemas' tree view is expanded to show the 'book_db' schema, which contains the 'authors' table. The 'Columns' section under 'authors' lists 'id', 'first_name', 'last_name', 'email', 'created_at', and 'updated_at'. The 'Tables' section also lists 'books', 'conversations', and 'messages'. The main pane displays the results of the query 'select * from authors;'. The result grid shows three rows of data:

	id	first_name	last_name	email	created_at	updated_at
1	1	vimalanandhan	sivanandham	vimal@gmail.com	2025-10-06T05:00:20	2025-10-06T05:11:58
2	2	George	Orwell	george.orwell@gmail.com	2025-10-05 21:07:54	2025-10-05 21:07:54
3	3	J.K.	Rowling	jk.rowling@gmail.com	2025-10-05 21:07:54	2025-10-05 21:07:54

AFTER DELETE:

The screenshot shows the MySQL Workbench interface after a delete operation. The 'Schemas' tree view remains the same. The main pane now displays the results of the query 'select * from authors;'. The result grid shows two rows of data, indicating that the row with id 3 has been deleted:

	id	first_name	last_name	email	created_at	updated_at
1	1	George	Orwell	george.orwell@gmail.com	2025-10-05 21:07:54	2025-10-05 21:07:54
2	2	J.K.	Rowling	jk.rowling@gmail.com	2025-10-05 21:07:54	2025-10-05 21:07:54

Books API Endpoint

6. POST /library/books - Create new book

POST /library/books Create Book

Create a new book

Parameters

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	<p>Response body</p> <pre>{ "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 } }</pre> <p>Download</p> <p>Response headers</p> <pre>access-control-allow-credentials: true content-length: 324 content-type: application/json date: Mon, 06 Oct 2025 05:53:57 GMT server: uvicorn</pre>

Responses

Code	Description
201	

Create a new book

Parameters

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
}
```

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
}'
```

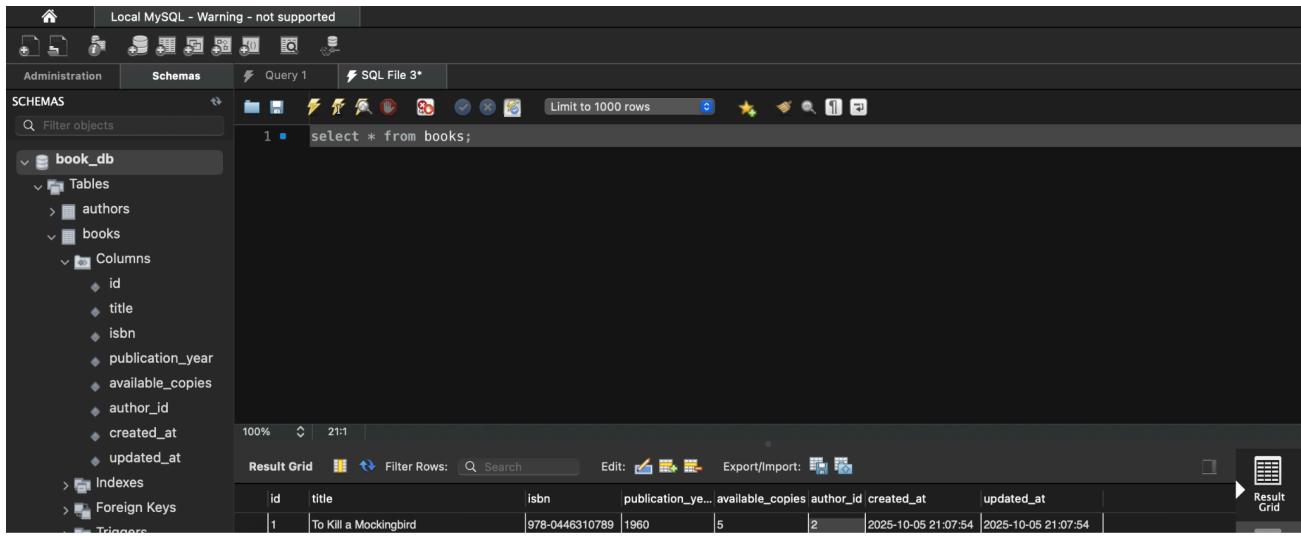
Request URL

http://localhost:8000/library/books

Server response

Code	Details	Links
201	<p>Response body</p> <pre>{ "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 } }</pre> <p>Response headers</p> <pre>access-control-allow-credentials: true content-length: 324 content-type: application/json date: Mon, 06 Oct 2025 05:53:57 GMT server: uvicorn</pre>	
Responses		
Code	Description	Links
201	Successful Response	No links

SQL:



The screenshot shows the MySQL Workbench interface. The left sidebar displays the schema structure for the 'book_db' database, including tables like 'authors' and 'books', and their columns such as 'id', 'title', 'isbn', and 'publication_year'. The main area shows the results of the SQL query 'select * from books;'. The result grid contains one row of data for the book 'To Kill a Mockingbird'.

	id	title	isbn	publication_ye...	available_copies	author_id	created_at	updated_at
1	1	To Kill a Mockingbird	978-0446310789	1960	5	2	2025-10-05 21:07:54	2025-10-05 21:07:54

7.GET /library/books - Get all books

GET /library/books Get Books

Get all books with pagination

Parameters

Name	Description
page	Page number
integer (query)	1
minimum: 1	
per_page	Items per page
integer (query)	10
maximum: 100	
minimum: 1	

Cancel

Execute Clear

Responses

Curl

```
curl -X 'GET' \
'http://localhost:8000/library/books?page=1&per_page=10' \
-H 'accept: application/json'
```

Request URL

http://localhost:8000/library/books?page=1&per_page=10

Server response

Code Details

200 Response body

```
{
  "books": [
    {
      "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
      }
    ],
    "total": 1,
    "page": 1,
    "per_page": 10,
    "total_pages": 1
}
```

Download

Response headers

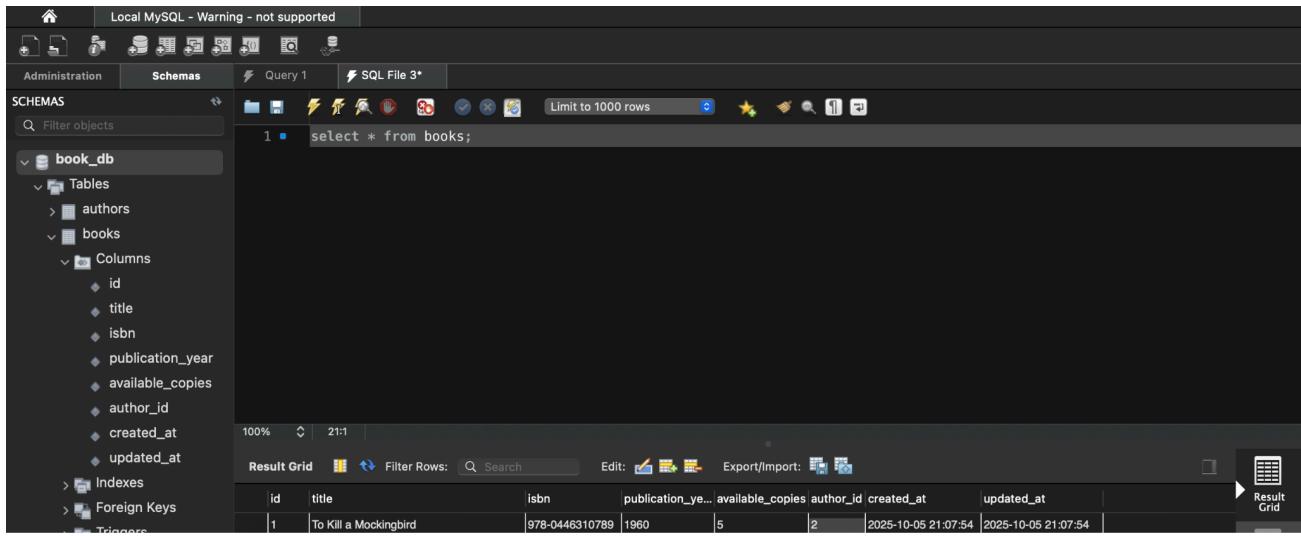
```
content-length: 385
content-type: application/json
date: Mon, 06 Oct 2025 05:54:04 GMT
server: uvicorn
```

Responses

Code Description Links

200 Successful Response No links

SQL:



The screenshot shows the MySQL Workbench interface. The left sidebar displays the schema structure for the 'book_db' database, including tables like 'authors' and 'books', and their columns such as 'id', 'title', 'isbn', and 'publication_year'. The main area shows the results of the SQL query:

```
1 • select * from books;
```

	id	title	isbn	publication_ye...	available_copies	author_id	created_at	updated_at
1	1	To Kill a Mockingbird	978-0446310789	1960	5	2	2025-10-05 21:07:54	2025-10-05 21:07:54

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

GET /library/books/{book_id} Get Book

Get a specific book by ID

Parameters

Name	Description
book_id * required integer (path)	1

Cancel

Execute Clear

Responses

Curl

```
curl -X 'GET' \
'http://localhost:8000/library/books/1' \
-H 'accept: application/json'
```

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": 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

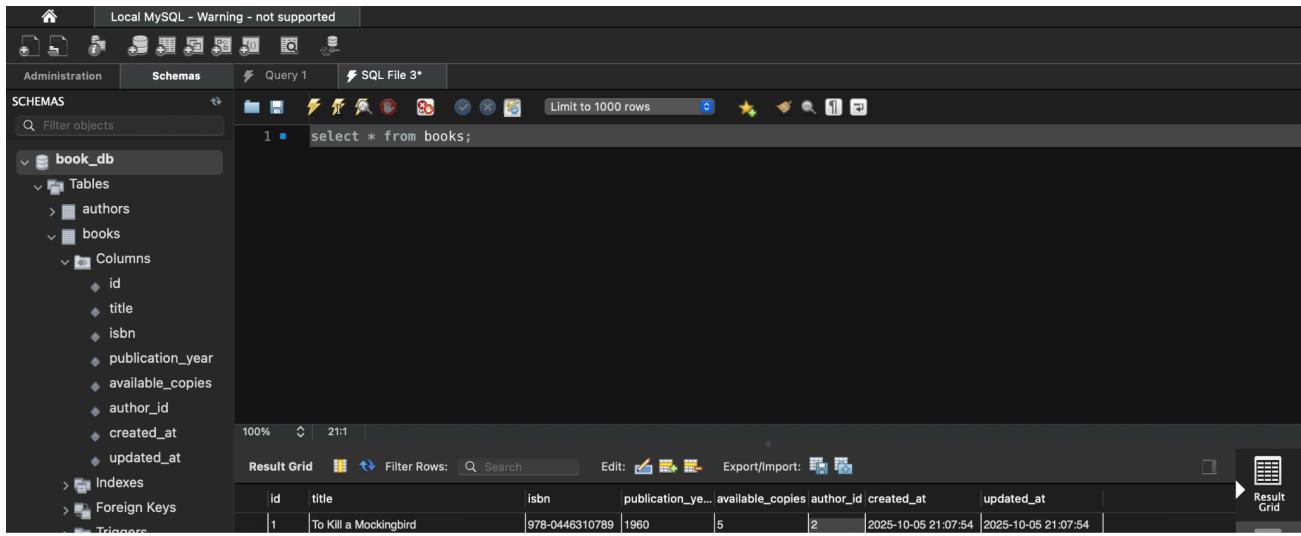
Response headers

```
content-length: 324
content-type: application/json
date: Mon, 06 Oct 2025 05:59:49 GMT
server: uvicorn
```

Responses

Code	Description	Links
200	Successful Response	No links

SQL:



The screenshot shows the MySQL Workbench interface. The left sidebar displays the schema structure for the 'book_db' database, including tables like 'authors' and 'books', and their columns such as 'id', 'title', 'isbn', and 'publication_year'. The main area shows the results of the SQL query 'select * from books;'. The result grid contains one row of data for the book 'To Kill a Mockingbird'.

	id	title	isbn	publication_ye...	available_copies	author_id	created_at	updated_at
1	1	To Kill a Mockingbird	978-0446310789	1960	5	2	2025-10-05 21:07:54	2025-10-05 21:07:54

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

GET /library/authors/{author_id}/books Get Books By Author

Get all books written by a specific author

Parameters

Name Description

author_id * required integer (path) 2

Cancel

Execute Clear

Responses

Curl

```
curl -X 'GET' \
'http://localhost:8000/library/authors/2/books' \
-H 'accept: application/json'
```

Request URL

<http://localhost:8000/library/authors/2/books>

Server response

Code Details

200 Response body

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

Download

Response headers

```
content-length: 326
content-type: application/json
date: Mon, 06 Oct 2025 06:02:44 GMT
server: unicorn
```

Responses

Code	Description	Links
200	Successful Response	No links

SQL:

The screenshot shows the MySQL Workbench interface. The left sidebar displays the database structure for 'book_db', including tables like 'authors', 'books', and 'columns'. The 'books' table is selected, showing columns: id, title, isbn, publication_year, available_copies, author_id, created_at, and updated_at. The main pane shows the results of the query 'select * from books;'. The results grid displays four rows of data:

	id	title	isbn	publication_ye...	available_copies	author_id	created_at	updated_at
1	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

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

PUT /library/books/{book_id} Update Book

Update a book's information

Parameters

Name	Description
book_id <small>required</small>	integer (path) 1

Request body required

application/json

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

Edit Value | Schema

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": "978-0446310799",
  "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: unicorn
```

Name	Description
book_id * required integer (path)	<input type="text" value="1"/>

Request body required

[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
  }'
```

[Copy](#)

Request URL

<http://localhost:8000/library/books/1>

Server response

Code	Details
200	<p>Response body</p> <pre>{ "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 } }</pre> <p>Copy Download</p> <p>Response headers</p> <pre>access-control-allow-credentials: true content-length: 342 content-type: application/json date: Mon, 06 Oct 2025 06:08:32 GMT server: uvicorn</pre>

Responses

Code	Description	Links
200	Successful Response	No links

SQL:

The screenshot shows the MySQL Workbench interface. The left sidebar displays the 'book_db' schema structure, including tables like authors, books, and columns such as id, title, isbn, publication_year, available_copies, author_id, created_at, and updated_at. The main area shows a query editor with the command 'select * from books;' and a results grid displaying the following data:

	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

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

DELETE /library/books/{book_id} Delete Book ^

Delete a book

Parameters

Name **Description**

book_id * required
integer
(path)

Cancel

Execute **Clear**

Responses

Curl

```
curl -X 'DELETE' \
  'http://localhost:8000/library/books/1' \
  -H 'accept: */*' 📋
```

Request URL

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

Server response

Code **Details**

204 Response headers

```
access-control-allow-credentials: true
content-type: application/json
date: Mon, 06 Oct 2025 06:28:18 GMT
server: unicorn
```

Responses

Code	Description	Links
204	Successful Response	No links

12. Test Duplicate Email

POST /library/authors Create Author

Create a new author

Parameters

No parameters

Request body required

application/json

Edit Value | Schema

```
{  
  "first_name": "vimal",  
  "last_name": "siva",  
  "email": "harper.lee@gmail.com"  
}
```

Execute Clear

Responses

Curl

```
curl -X 'POST' \  
  'http://localhost:8000/library/authors' \  
  -H 'accept: application/json' \  
  -H 'Content-Type: application/json' \  
  -d '{  
    "first_name": "string",  
    "last_name": "string",  
    "email": "harper.lee@gmail.com"  
}'
```

Request URL

http://localhost:8000/library/authors

Server response

Code Details

[Edit Value](#) | Schema

```
{  
  "first_name": "vimal",  
  "last_name": "siva",  
  "email": "harper.lee@gmail.com"  
}
```

[Execute](#)

[Clear](#)

Responses

Curl

```
curl -X 'POST' \  
  'http://localhost:8000/library/authors' \  
  -H 'accept: application/json' \  
  -H 'Content-Type: application/json' \  
  -d '{  
    "first_name": "string",  
    "last_name": "string",  
    "email": "harper.lee@gmail.com"  
}'
```



Request URL

<http://localhost:8000/library/authors>

Server response

Code **Details**

400 Error: Bad Request
Undocumented

Response body

```
{  
  "detail": "Email already exists"  
}
```



[Download](#)

Response headers

```
access-control-allow-credentials: true  
content-length: 33  
content-type: application/json  
date: Mon, 06 Oct 2025 06:12:30 GMT  
server: uvicorn
```

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:

The screenshot shows the MySQL Workbench interface. On the left, the 'Schemas' tree view is expanded to show the 'book_db' schema, which contains tables like 'authors' and 'books'. The 'Tables' section under 'book_db' is selected. In the center, the 'Query 1' tab contains the SQL command: 'select * from authors;'. Below the query, the 'Result Grid' displays the data from the 'authors' table:

	id	first_name	last_name	email	created_at	updated_at
1	1	Harper	Lee	harper.lee@gmail.com	2025-10-05 21:07:54	2025-10-05 21:07:54
2	2	George	Orwell	george.orwell@gmail.com	2025-10-05 21:07:54	2025-10-05 21:07:54
3	3	J.K.	Rowling	jk.rowling@gmail.com	2025-10-05 21:07:54	2025-10-05 21:07:54
	NULL	NULL	NULL	NULL	NULL	NULL

13. Test Duplicate ISBN

POST /library/books Create Book

Create a new book

Parameters

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

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
400 <i>Undocumented</i>	Error: Bad Request Response body <pre>{ "detail": "ISBN already exists" }</pre> Response headers <pre>access-control-allow-credentials: true content-length: 32 content-type: application/json date: Mon, 06 Oct 2025 06:16:51 GMT server: uvicorn</pre>

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:

The screenshot shows the MySQL Workbench interface with the following details:

- Left Panel (Schemas):** The "book_db" schema is selected. Under "Tables", the "books" table is expanded, showing columns: id, title, isbn, publication_year, available_copies, author_id, created_at, and updated_at.
- Top Bar:** The title bar says "Local MySQL - Warning - not supported". The tabs "Query 1" and "SQL File 3*" are visible. A search bar contains the query "select * from books;".
- Result Grid:** The results of the query are displayed in a grid format. The columns are id, title, isbn, publication_ye..., available_copies, author_id, created_at, and updated_at. The data rows are:

	id	title	isbn	publication_ye...	available_copies	author_id	created_at	updated_at
1	1	To Kill a Mockingbird	978-0446310789	1960	5	1	2025-10-05 21:07:54	2025-10-05 21:07:54
2	2	1984	978-0451524935	1949	3	2	2025-10-05 21:07:54	2025-10-05 21:07:54
3	3	Animal Farm	978-0451526342	1945	2	2	2025-10-05 21:07:54	2025-10-05 21:07:54
4	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
- Right Panel:** Buttons for "Result Grid", "Form Editor", and "Breadcrumbs" are visible.

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

The screenshot shows a REST API testing interface. At the top, a red button labeled "DELETE" is followed by the URL "/library/authors/{author_id}" and the description "Delete Author". Below this is a pink header area with the text "Delete an author (only if they have no associated books)".

The main section is titled "Parameters" and contains a table:

Name	Description
author_id * required integer (path)	2

Below the parameters are two buttons: "Execute" (blue) and "Clear" (grey).

The "Responses" section starts with "Responses" and includes a "Curl" block containing the command:

```
curl -X 'DELETE' \
  'http://localhost:8000/library/authors/2' \
  -H 'accept: */*'
```

It also includes a "Request URL" block with the value "http://localhost:8000/library/authors/2".

The "Server response" section shows a "Code" of 400 and a "Details" of "Error: Bad Request".

The "Response body" block displays the JSON object:

```
{  
  "detail": "Cannot delete author. They have 1 associated book(s). Please delete the books first."  
}
```

There are "Copy" and "Download" buttons next to this block.

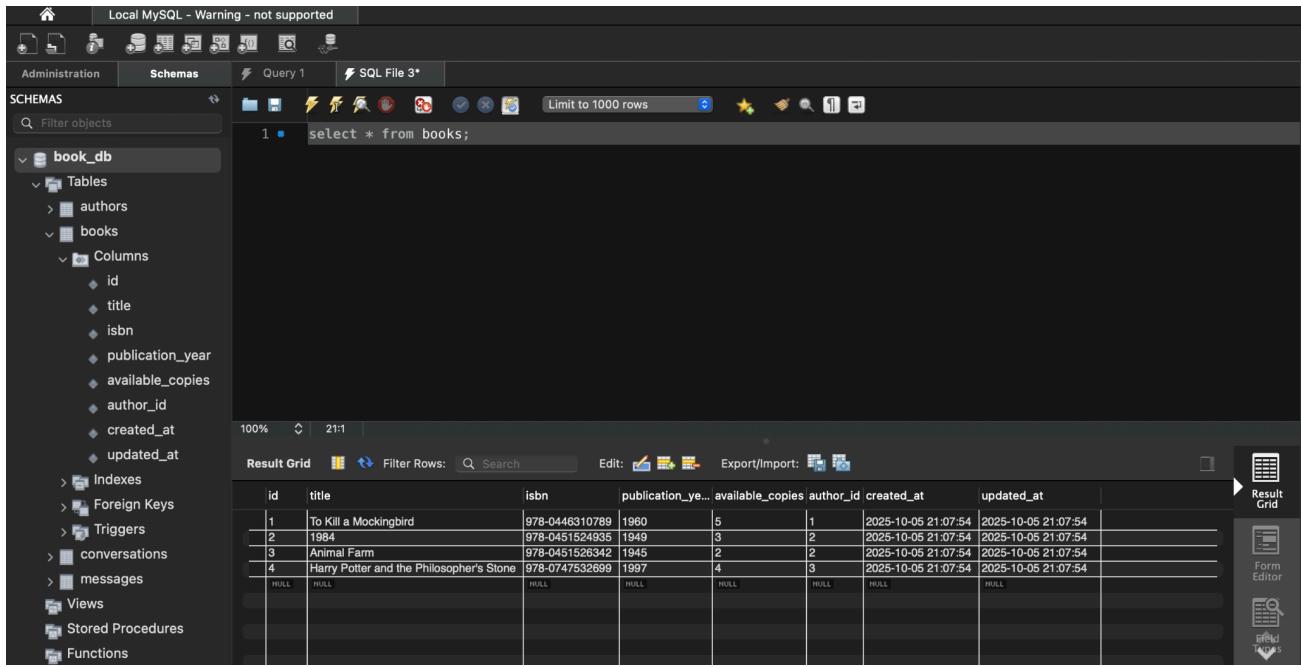
The "Response headers" block shows the following headers:

```
access-control-allow-credentials: true  
content-length: 97  
content-type: application/json  
date: Mon, 06 Oct 2025 06:21:34 GMT  
server: uvicorn
```

A "Responses" button is located at the bottom left of the responses section.

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:



The screenshot shows the MySQL Workbench interface with the following details:

- Toolbar:** Includes icons for Home, Database, Schemas, Tables, Columns, Triggers, Functions, and various search and export tools.
- Schemas Tab:** Shows the current schema is "book_db".
- Tables Tab:** Shows tables "authors" and "books".
- Columns Tab:** Shows columns for the "books" table: id, title, isbn, publication_year, available_copies, author_id, created_at, and updated_at.
- Triggers Tab:** Shows triggers "1984" and "Harry Potter and the Philosopher's Stone".
- Functions Tab:** Shows functions "Animal Farm".
- Result Grid:** Displays the output of the query "select * from books;".
- Query Editor:** Contains the SQL command "select * from books;".
- Table Headers:** The result grid has columns: id, title, isbn, publication_ye..., available_copies, author_id, created_at, and updated_at.
- Data Rows:** Four rows of data are shown:

	id	title	isbn	publication_ye...	available_copies	author_id	created_at	updated_at
1	1	To Kill a Mockingbird	978-0446310789	1960	5	1	2025-10-05 21:07:54	2025-10-05 21:07:54
2	2	1984	978-0451524935	1949	3	2	2025-10-05 21:07:54	2025-10-05 21:07:54
3	3	Animal Farm	978-0451526342	1945	2	2	2025-10-05 21:07:54	2025-10-05 21:07:54
4	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

SQL TABLES:

The screenshot shows a database structure for a table named 'authors' within a database named 'book_db'. The table has the following columns:

- id**
- first_name**
- last_name**
- email**
- created_at**
- updated_at**

Indexes defined for the table are:

- PRIMARY**
- email**
- idx_email**

Foreign Keys and Triggers are also listed.

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.

The image shows a file explorer interface with a dark theme. At the top, there is a header with the text "HW 4" and several small icons for file operations like copy, paste, and search. Below the header, the file structure is listed:

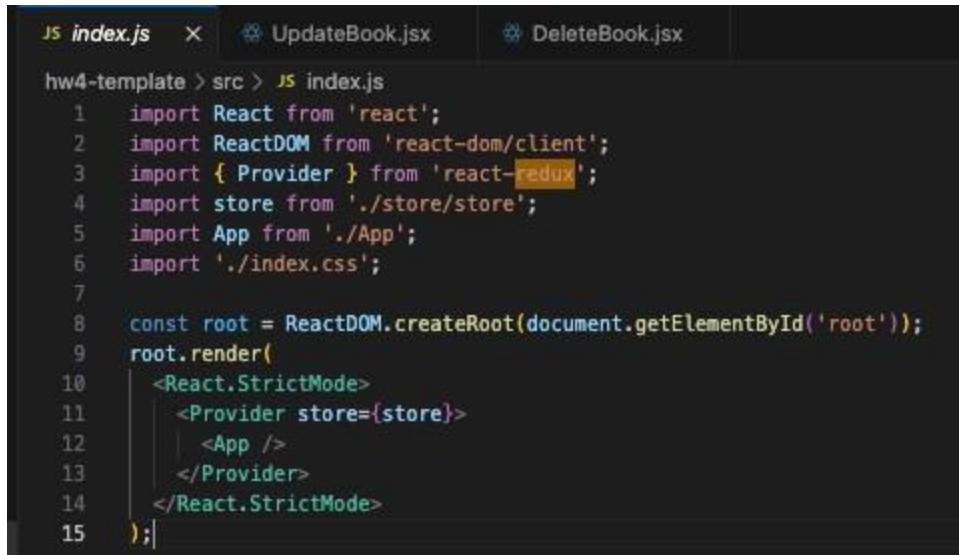
- hw4-template
 - node_modules
 - public
 - src
 - components
 - Create
 - Delete
 - Home
 - Update
 - services
 - JS api.js
 - store
 - JS bookSlice.js
 - JS store.js
 - # App.css
 - JS App.js
 - # index.css
 - JS index.js
 - JS setupTests.js
 - .gitignore
 - { package-lock.json
 - { package.json
 -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

The screenshot shows a file explorer window with the following directory structure:

- 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

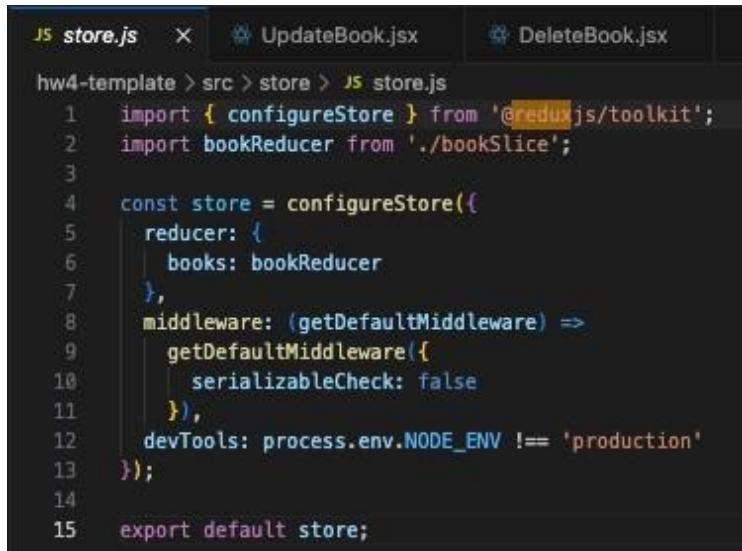
```
>CreateBook.jsx    JS api.js      X  UpdateBook.jsx  DeleteBook.jsx
hw4-template > src > services > JS api.js
1 import axios from 'axios';
2
3 const api = axios.create({
4   baseURL: 'http://localhost:3000/api'
5 });
6
7 // Request interceptor for error handling
8 api.interceptors.request.use(
9   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



```
JS index.js  X  ⚡ UpdateBook.jsx  ⚡ DeleteBook.jsx

hw4-template > src > JS index.js
1 import React from 'react';
2 import ReactDOM from 'react-dom/client';
3 import { Provider } from 'react-redux';
4 import store from './store/store';
5 import App from './App';
6 import './index.css';
7
8 const root = ReactDOM.createRoot(document.getElementById('root'));
9 root.render(
10   <React.StrictMode>
11     <Provider store={store}>
12       <App />
13     </Provider>
14   </React.StrictMode>
15 );
```



```
JS store.js  X  ⚡ UpdateBook.jsx  ⚡ DeleteBook.jsx

hw4-template > src > store > JS store.js
1 import { configureStore } from '@reduxjs/toolkit';
2 import bookReducer from './bookSlice';
3
4 const store = configureStore({
5   reducer: {
6     books: bookReducer
7   },
8   middleware: getDefaultMiddleware =>
9     getDefaultMiddleware({
10       serializableCheck: false
11     }),
12   devTools: process.env.NODE_ENV !== 'production'
13 });
14
15 export default store;
```

```
JS bookSlice.js ✘  ⚡ UpdateBook.jsx  ⚡ DeleteBook.jsx
hw4-template > src > store > JS bookSlice.js
1 import { createSlice, createAsyncThunk } from '@reduxjs/toolkit
2 import * as api from '../services/api';
3
4 // Async Thunks
5 export const fetchBooks = createAsyncThunk(
6   'books/fetchBooks',
7   async () => {
8     return await api.getBooks();
9   }
10 );
11
12 export const createBook = createAsyncThunk(
13   'books/createBook',
14   async (bookData) => {
15     return await api.createBook(bookData);
16   }
17 );
18
19 export const updateBook = createAsyncThunk(
20   'books/updateBook',
21   async ({ id, bookData }) => {
22     return await api.updateBook(id, bookData);
23   }
24 );
25
26 export const deleteBook = createAsyncThunk(
27   'books/deleteBook',
28   async (id) => [
29     await api.deleteBook(id);
30     return id;
31   ]
32 );
33
34 const bookSlice = createSlice({
35   name: 'books',
36   initialState: {
37     books: [],
38     loading: false,
39     error: null
40   },
41   reducers: {
42     clearError: (state) => {
43       state.error = null;
44     }
45   },
46   extraReducers: (builder) => {
47     builder
48       // Fetch Books
49       .addCase(fetchBooks.pending, (state) => {
50         state.loading = true;
51         state.error = null;
52       })
53       .addCase(fetchBooks.fulfilled, (state, action) => {
54         state.loading = false;
55         state.books = action.payload;
56       })
57       .addCase(fetchBooks.rejected, (state, action) => {
58         state.loading = false;
59         state.error = action.error.message;
60       })
61   }
62 }
```

```
JS bookSlice.js ✘ UpdateBook.jsx ✘ DeleteBook.jsx
hw4-template > src > store > JS bookSlice.js
  ...
61      // Create Book
62      .addCase(createBook.pending, (state) => {
63        state.loading = true;
64        state.error = null;
65      })
66      .addCase(createBook.fulfilled, (state, action) => {
67        state.loading = false;
68        state.books.push(action.payload);
69      })
70      .addCase(createBook.rejected, (state, action) => {
71        state.loading = false;
72        state.error = action.error.message;
73      })
74    // Update Book
75    .addCase(updateBook.pending, (state) => {
76      state.loading = true;
77      state.error = null;
78    })
79    .addCase(updateBook.fulfilled, (state, action) => {
80      state.loading = false;
81      state.books = state.books.map(book =>
82        book.id === action.payload.id ? action.payload : book
83      );
84    })
85    .addCase(updateBook.rejected, (state, action) => {
86      state.loading = false;
87      state.error = action.error.message;
88    })
89  // Delete Book
90  .addCase(deleteBook.pending, (state) => {
91    state.loading = true;
92    state.error = null;
93  })
94  .addCase(deleteBook.fulfilled, (state, action) => {
95    state.loading = false;
96    state.books = state.books.filter(book => book.id !== action.payload);
97  })
98  .addCase(deleteBook.rejected, (state, action) => {
99    state.loading = false;
100   state.error = action.error.message;
101 });
102 }
103 );
104
105 export const { clearError } = bookSlice.actions;
106 export default bookSlice.reducer;
```



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

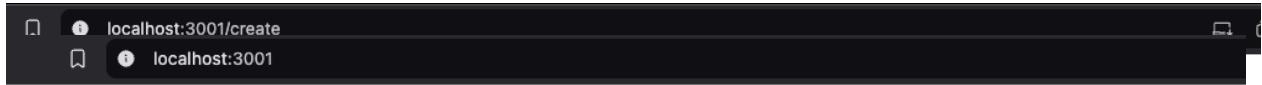
Home.jsx X

UpdateBook.jsx

DeleteBook.jsx

```
hw4-template > src > components > Home > Home.jsx
1 import React, { useEffect } from 'react';
2 import { Link } from 'react-router-dom';
3 import { useDispatch, useSelector } from 'react-redux';
4 import { fetchBooks, deleteBook } from '../../../../../store/bookSlice';
5 import './Home.css';
6
7 const Home = () => {
8   const dispatch = useDispatch();
9   const { books, loading, error } = useSelector(state => state.books);
10
11  useEffect(() => {
12    dispatch(fetchBooks());
13  }, [dispatch]);
14
15  const handleDelete = async (id) => {
16    try {
17      await dispatch(deleteBook(id)).unwrap();
18    } catch (err) {
19      console.error('Failed to delete book:', err);
20    }
21  };
22
23  if (loading) return <div>Loading...</div>;
24  if (error) return <div>Error: {error}</div>;
25
26  return (
27    <div className="home-container">
28      <h1>Book Management System</h1>
29      <Link to="/create" className="add-button">Add New Book</Link>
30
31      <div className="books-grid">
32        {books.map(book => (
33          <div key={book.id} className="book-card">
34            <h3>{book.title}</h3>
35            <p>Author: {book.author}</p>
36            <p>ISBN: {book.isbn}</p>
37            <div className="book-actions">
38              <Link to={`/update/${book.id}`} className="edit-button">Update</Link>
39              <button
40                onClick={() => handleDelete(book.id)}
41                className="delete-button"
42              >
43                Delete
44              </button>
45            </div>
46          </div>
47        )));
48      </div>
49    );
50  };
51
52  export default Home;
```

Create Screen



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

Author: Test

ISBN: 12345654321

[Update](#)

[Delete](#)

The screenshot shows the Chrome DevTools interface with the Redux extension active. The top navigation bar includes Elements, Console, Sources, Redux (which is selected), Network, Performance, Memory, Application, Privacy and security, and a few more items on the right. Below the navigation is a toolbar with Actions (refresh, lock, reset, revert, sweep, commit) and Settings. A status bar at the top right shows the ID 1068381183/1.

The main area is divided into two sections: a left sidebar and a right panel. The sidebar contains a 'filter...' input field and a list of actions:

- @@INIT (timestamp: 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)

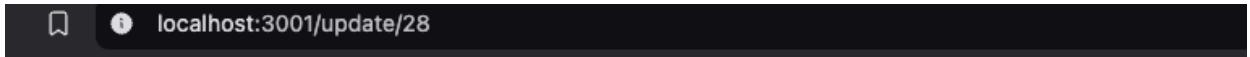
The right panel is titled 'Diff' and contains tabs for Action, State, Diff (selected), Trace, and Test. It shows a hierarchical tree view of state changes:

- Tree**: Shows the structure of state changes.
 - 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
 - Raw**: Shows the raw JSON representation of the state changes.

```
>CreateBook.jsx UpdateBook.jsx DeleteBook.jsx

hw4-template > src > components > Create > CreateBook.jsx
1 import React, { useState } from 'react';
2 import { useNavigate } from 'react-router-dom';
3 import { useDispatch, useSelector } from 'react-redux';
4 import { createBook } from '../../../../../store/bookSlice';
5 import './CreateBook.css';
6
7 const CreateBook = () => {
8     const [formData, setFormData] = useState({
9         title: '',
10        author: '',
11        isbn: '',
12        publishedYear: ''
13    });
14    const dispatch = useDispatch();
15    const navigate = useNavigate();
16    const { loading, error } = useSelector(state => state.books);
17
18    const handleChange = (e) => {
19        const { name, value } = e.target;
20        setFormData(prev => ({
21            ...prev,
22            [name]: value
23        }));
24    };
25
26    const handleSubmit = async (e) => {
27        e.preventDefault();
28        try {
29            await dispatch(createBook(formData)).unwrap();
30            navigate('/');
31        } catch (err) {
32            console.error('Failed to create book:', err);
33        }
34    };
35
36    return [
37        <div className="create-book-container">
38            <h2>Add New Book</h2>
39            {error && <div className="error-message">{error}</div>}
40            <form onSubmit={handleSubmit}>
41                <div className="form-group">
42                    <label htmlFor="title">Book Title:</label>
43                    <input
44                        type="text"

```



Update Book

Book Title:

Test - Updated

Author Name:

Test - Updated

ISBN:

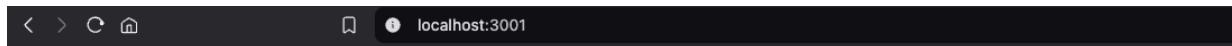
12345654321

Published Year:

1990

[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](#)

Test - Updated

Author: Test - Updated

ISBN: 12345654321

[Update](#)

[Delete](#)

Screenshot of the Redux DevTools extension in a browser's developer tools.

The top navigation bar includes: Elements, Console, Sources, **Redux**, Network, Performance, Memory, Application, Privacy and security, and various icons for Actions, Settings, Reset, Revert, Sweep, Commit, and a search bar.

The main area shows a list of actions on the left and a detailed diff view on the right.

Actions:

- books/receiveBooks/pending
- 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): <code>'Test'</code> => <code>'Test - Updated'</code>				
author (pin): <code>'Test'</code> => <code>'Test - Updated '</code>				
updatedAt (pin): <code>'2025-10-04T00:57:44.000Z'</code> => <code>'2025-10-04T01:01:21.517Z'</code>				
loading (pin): <code>true</code> => <code>false</code>				

books/fetchBooks/fulfilled (16)

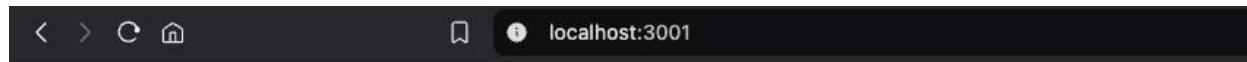
Control buttons at the bottom include: play, back, forward, Live, 1x, and 2x.

The screenshot shows the Chrome DevTools interface with the **Redux** tab selected. The left sidebar lists actions: `books/fetchBooks/pending`, `books/fetchBooks/pending`, `books/fetchBooks/fulfilled`, `books/fetchBooks/fulfilled`, `books/createBook/pending`, `books/createBook/fulfilled`, `books/fetchBooks/pending`, `books/fetchBooks/pending`, `books/fetchBooks/fulfilled`, `books/fetchBooks/fulfilled`, `books/fetchBooks/fulfilled`, `books/fetchBooks/fulfilled`, `books/fetchBooks/fulfilled`, `books/fetchBooks/fulfilled`, and `books/fetchBooks/fulfilled`. The right panel displays a state diff between two snapshots. The **Diff** table has columns for **Action**, **State**, **Diff**, **Trace**, and **Test**. The **Tree** tab is selected, showing a JSON structure of the state:

```
{  
  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  
    },  
  }  
}
```

```
>CreateBook.jsx  UpdateBook.jsx  DeleteBook.jsx
hw4-template > src > components > Update > UpdateBook.jsx
  1 import React, { useState, useEffect } from 'react';
  2 import { useNavigate, useParams } from 'react-router-dom';
  3 import { useDispatch, useSelector } from 'react-redux';
  4 import { updateBook } from '../../store/bookSlice';
  5 import { getBookById } from '../../services/api';
  6 import './UpdateBook.css';
  7
  8 const UpdateBook = () => {
  9   const [formData, setFormData] = useState({
 10     title: '',
 11     author: '',
 12     isbn: '',
 13     publishedYear: ''
 14   });
 15
 16   const navigate = useNavigate();
 17   const dispatch = useDispatch();
 18   const { id } = useParams();
 19   const { loading, error } = useSelector(state => state.books);
 20   const [fetchError, setFetchError] = useState(null);
 21
 22   useEffect(() => {
 23     const fetchBook = async () => {
 24       try {
 25         const book = await getBookById(id);
 26         setFormData(book);
 27       } catch (error) {
 28         setFetchError(error.message || 'Failed to fetch book');
 29       }
 30     };
 31
 32     fetchBook();
 33   }, [id]);
 34
 35   const handleChange = (e) => {
 36     const { name, value } = e.target;
 37     setFormData(prev => ({
 38       ...prev,
 39       [name]: value
 40     }));
 41   };
 42
 43   const handleSubmit = async (e) => {
 44     e.preventDefault();
 45     try {
 46       await dispatch(updateBook({ id, bookData: formData })).unwrap();
 47       navigate('/');
 48     } catch (err) {
 49       console.error('Failed to update book:', err);
 50     }
 51   };
 52
 53   if (loading) return <div>Loading...</div>;
 54   if (fetchError) return <div>Error: {fetchError}</div>;
 55
 56   return (
 57     <div className="update-book-container">
 58       <h2>Update Book</h2>
 59       {error && <div className="error-message">{error}</div>}
 60       <form onSubmit={handleSubmit}>
 61         <div className="form-group">
 62           <label htmlFor="title">Book Title:</label>
```

Delete Screen



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](#)

The screenshot shows the Chrome DevTools interface with the **Redux** panel open. The top navigation bar includes **Elements**, **Console**, **Sources**, **Redux** (which is selected), **Network**, **Performance**, **Memory**, **Application**, **Privacy and security**, and other developer tools.

The **Actions** tab is active, showing a list of actions taken:

- books/fetchBooks/pending
- 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 (+00:00.04)
- books/deleteBook/pending (Jump Skip)
- books/deleteBook/fulfilled (+00:00.04)

The **Diff** section displays the state difference between two snapshots. The **Tree** tab is selected, showing a hierarchical view of the state changes. The **Raw** tab is also present. The diff output is as follows:

```
{  
  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  
    ],  
  },  
}
```

At the bottom, there is a timeline slider labeled **books/fetchBooks/fulfilled (15)** with controls for **Jump** and **Skip**. To the right of the timeline are buttons for **Live**, **1x**, and **2x**.

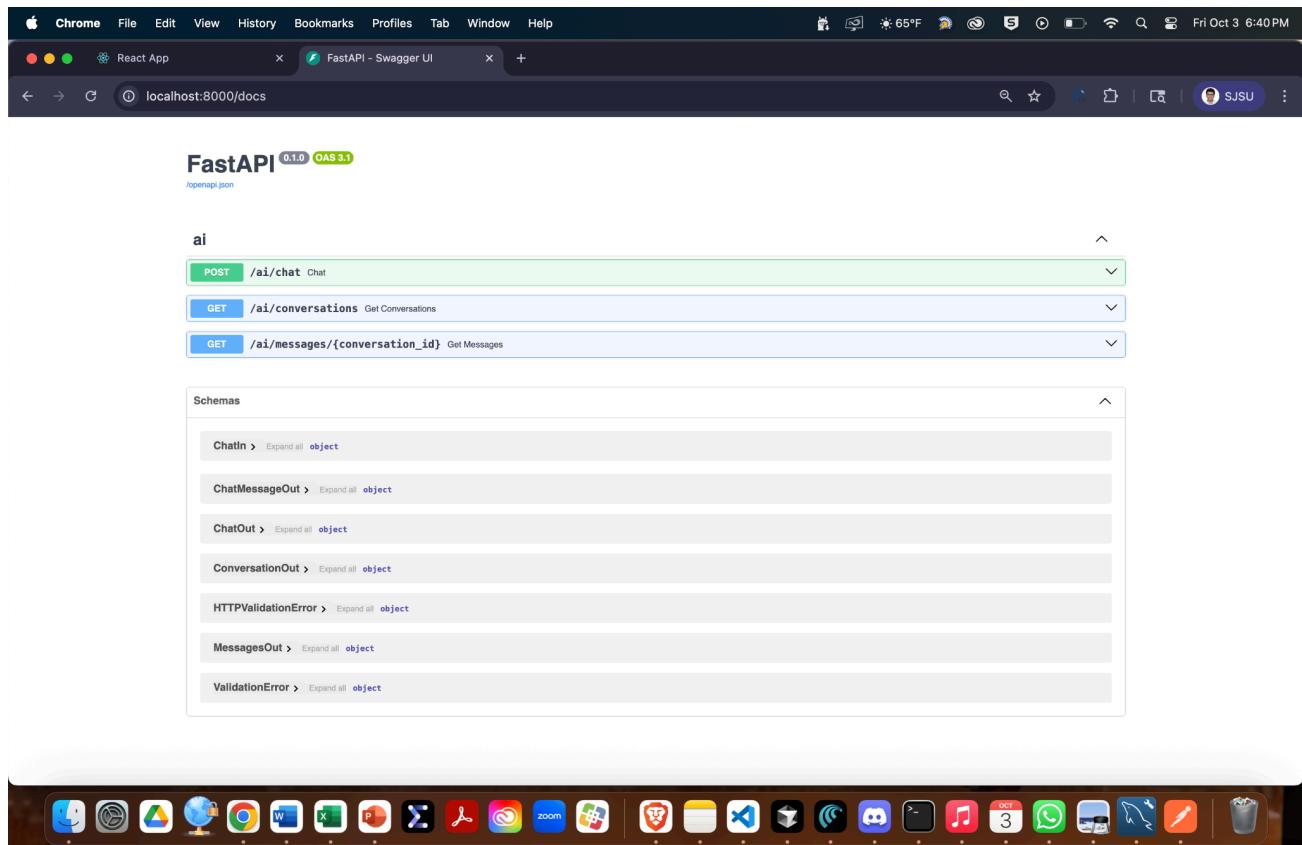
```
>CreateBook.jsx UpdateBook.jsx DeleteBook.jsx X
hw4-template > src > components > Delete > DeleteBook.jsx
1 import React, { useState, useEffect } from 'react';
2 import { useNavigate, useParams } from 'react-router-dom';
3 import { useDispatch, useSelector } from 'react-redux';
4 import { deleteBook } from '../../../../../store/bookSlice';
5 import { getBookById } from '../../../../../services/api';
6 import './DeleteBook.css';
7
8 const DeleteBook = () => {
9   const [book, setBook] = useState(null);
10  const [error, setError] = useState(null);
11  const navigate = useNavigate();
12  const dispatch = useDispatch();
13  const { id } = useParams();
14  const { loading } = useSelector(state => state.books);
15
16  useEffect(() => {
17    const fetchBook = async () => {
18      try {
19        const bookData = await getBookById(id);
20        setBook(bookData);
21      } catch (error) {
22        setError('Failed to fetch book details');
23        console.error('Error:', error);
24      }
25    };
26
27    fetchBook();
28  }, [id]);
29
30  const handleDelete = async () => {
31    try {
32      await dispatch(deleteBook(id)).unwrap();
33      navigate('/');
34    } catch (error) {
35      setError(error.message);
36      console.error('Error:', error);
37    }
38  };
39
40  if (loading) return <div>Loading...</div>;
41  if (error) return <div className="error-message">{error}</div>;
42
43  return (
44    <div className="delete-book-container">
45      <h2>Delete Book</h2>
46      {book && (
47        <div className="book-details">
48          <h3>{book.title}</h3>
49          <p>Author: {book.author}</p>
50          <p>ISBN: {book.isbn}</p>
51          {book.publishedYear && <p>Published Year: {book.publishedYear}</p>}
52          <p className="warning-text">Are you sure you want to delete this book?</p>
53        </div>
54      )}

```

PART -3

LLM Chat (Ollama + FastAPI + React)

API responses in Swagger/Postman:



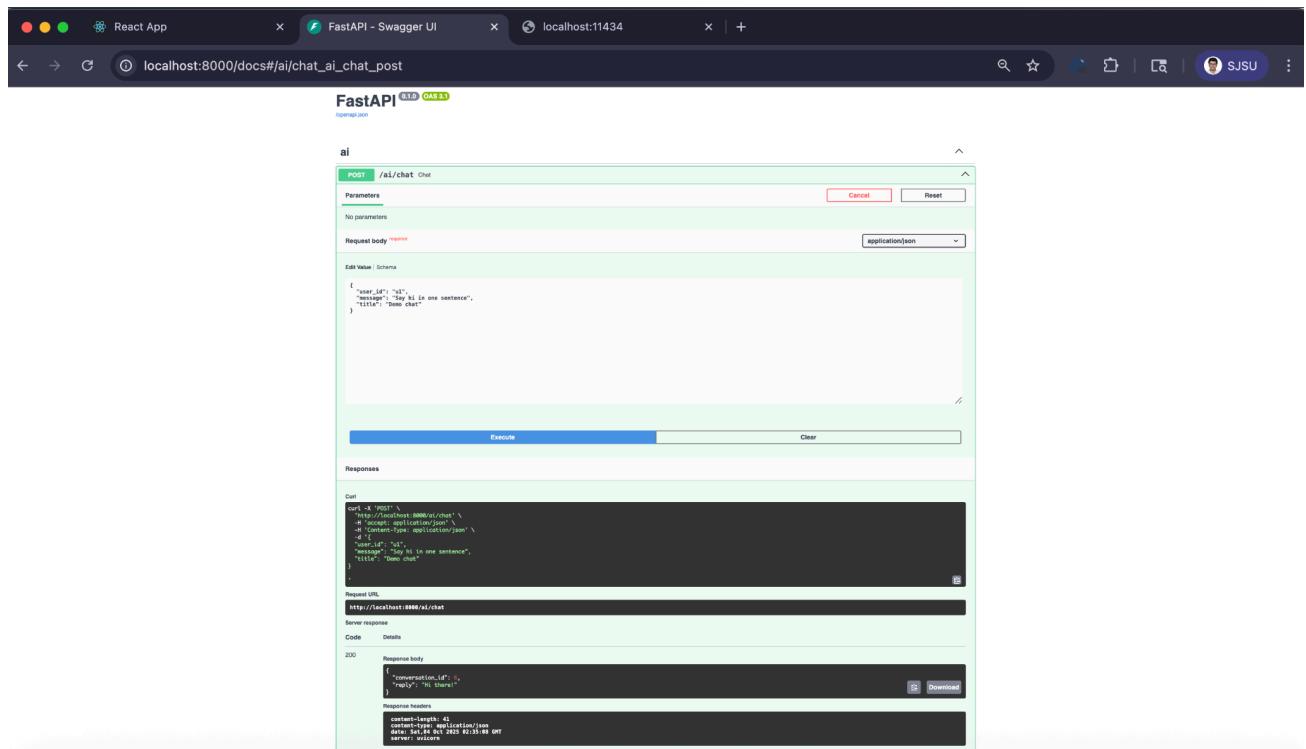
The screenshot shows a browser window with multiple tabs open. The active tab is titled "LLM Chat API - Swagger UI". The URL in the address bar is "127.0.0.1:8000/docs#/".

The main content area displays the "Schemas" section of the Swagger UI. It lists several API models:

- ChatIn**:
A schema object with properties:
 - `user_id`: string
 - `message`: string
 - `conversation_id`: integer or null
 - `title`: string or null
- ChatMessageOut**:
A schema object with properties:
 - `id`: integer
 - `role`: string
 - `content`: string
 - `created_at`: string (date-time)
- ChatOut**:
A schema object with properties:
 - `conversation_id`: integer
 - `reply`: string
- ConversationOut**:
A schema object with properties:
 - `user_id`: string
 - `title`: string or null
 - `AnyOf`:
 - string or null
 - array of strings or null
 - `created_at`: string (date-time)
 - `updated_at`: string (date-time)
- HTTPValidationError**:
A schema object with properties:
 - `details`: array of objects
 - `Items`:
 - `lack`: object of arrays of strings or integers
 - `Items`:
 - `id`: string or integer
 - `AnyOf`:
 - string
 - integer
 - `msg`: string
 - `type`: string
- MessageOut**:
A schema object with properties:
 - `conversation_id`: integer
 - `messages`: array of objects
 - `Items`:
 - `id`: integer
 - `role`: string
 - `content`: string
 - `created_at`: string (date-time)
- ValidationError**:
A schema object with properties:
 - `lack`: array of strings or integers
 - `Items`:
 - `id`: string or integer
 - `AnyOf`:
 - string
 - integer
 - `msg`: string
 - `type`: string

1) POST /ai/chat — start a conversation

Swagger:



The screenshot shows the FastAPI Swagger UI interface. At the top, it displays the URL `localhost:8000/docs#/ai/chat_ai_chat_post`. Below this, the `POST /ai/chat Chat` endpoint is selected. The `Request body` section shows the following schema:

```
{ "user_id": "ai", "message": "Say hi in one sentence", "title": "Demo chat" }
```

The `Responses` section shows the following details for a 200 status code:

- `Curl`: curl -X POST -H "Content-Type: application/json" -d '{"user_id": "ai", "message": "Say hi in one sentence", "title": "Demo chat"}'
- `Request URL`: http://localhost:8000/ai/chat
- `Server response`:
 - `Code`: 200
 - `Response body`:

```
{ "conversation_id": 1, "reply": "Hi there!" }
```
 - `Response headers`:

```
content-length: 41
content-type: application/json
date: Sat, 09 Oct 2022 02:35:09 GMT
server: uvicorn
```

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

The screenshot shows the FastAPI Swagger UI interface for a POST endpoint named `/ai/chat`. The request body schema is defined as follows:

```
POST /ai/chat
{
    "parameters": [
        {
            "name": "user_id",
            "in": "query",
            "type": "string"
        },
        {
            "name": "message",
            "in": "query",
            "type": "string"
        },
        {
            "name": "conversation_id",
            "in": "query",
            "type": "integer"
        }
    ],
    "requestBody": {
        "content": {
            "application/json": {
                "schema": {
                    "type": "object",
                    "properties": {
                        "user_id": {"type": "string", "format": "string"}, // This is a string, not a query parameter
                        "message": {"type": "string", "format": "string"}, // This is a string, not a query parameter
                        "conversation_id": {"type": "integer", "format": "int32"} // This is an integer, not a query parameter
                    }
                }
            }
        }
    }
}
```

The response section shows a successful 200 OK status with the following JSON body:

```
HTTP/1.1 200 OK
Content-Type: application/json
Date: Mon, 24 Oct 2022 02:28:24 UTC
Server: uvicorn
{
    "conversation_id": 1,
    "reply": "The shortest war in history was between Britain and Zanzibar on August 27, 1896, and lasted only 38 minutes!"
}
```

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

Swagger: set user_id = u1 in the query param

FastAPI 0.1.0 OAS 3.1
[/openapi.json](#)

ai

POST /ai/chat Chat

GET /ai/conversations Get Conversations

Parameters

Name	Description
user_id <small>required</small>	string (query) u1

Execute Clear

Responses

Curl

```
curl -X 'GET' \
  'http://localhost:8000/ai/conversations?user_id=u1' \
  -H 'accept: application/json'
```

Request URL

http://localhost:8000/ai/conversations?user_id=u1

Server response

Code Details

200 Response body

```
[  
  {  
    "id": 3,  
    "user_id": "u1",  
    "title": "Test chat",  
    "created_at": "2025-10-03T19:10:52",  
    "updated_at": "2025-10-03T19:10:52"  
  },  
  {  
    "id": 4,  
    "user_id": "u1",  
    "title": "Test Chat",  
    "created_at": "2025-10-03T19:23:58",  
    "updated_at": "2025-10-03T19:23:58"  
  },  
  {  
    "id": 6,  
    "user_id": "u1",  
    "title": "Demo chat",  
    "created_at": "2025-10-03T19:35:09",  
    "updated_at": "2025-10-03T19:35:09"  
  }]
```

Download

Response headers

```
content-length: 343  
content-type: application/json  
date: Sat, 04 Oct 2025 02:39:14 GMT  
server: uvicorn
```

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

Swagger: path param conversation_id = 1

FastAPI 0.1.0 OAS 3.1
[/openapi.json](#)

ai

POST /ai/chat Chat

GET /ai/conversations Get Conversations

GET /ai/messages/{conversation_id} Get Messages

Parameters

Name Description

conversation_id * required
integer
(path)
1

Execute Clear

Responses

Curl

```
curl -X 'GET' \
  'http://localhost:8000/ai/messages/1' \
  -H 'accept: application/json'
```

Request URL

```
http://localhost:8000/ai/messages/1
```

Server response

Code Details

200 Response body

```
{
  "messages": [
    {
      "id": 1,
      "role": "user",
      "content": "Hello! Vimal",
      "created_at": "2025-10-03T18:44:32"
    },
    {
      "id": 8,
      "role": "user",
      "content": "Give me a fun fact in one sentence.",
      "created_at": "2025-10-03T19:28:26"
    },
    {
      "id": 9,
      "role": "assistant",
      "content": "The shortest war in history was between Britain and Zanzibar on August 27, 1896, and lasted only 38 minutes!",
      "created_at": "2025-10-03T19:28:40"
    }
  ]
}
```

Download

Response headers

```
content-length: 387
content-type: application/json
date: Sat, 04 Oct 2025 02:41:03 GMT
server: unicorn
```

Responses

Code Description Links

200 Successful Response No links

DB Screenshots:

Conversations list

The screenshot shows the MySQL Workbench interface with the following details:

- Left Panel (Schemas):** The "book_db" schema is selected. Under "Tables", the "conversations" table is expanded, showing its columns: id, user_id, title, created_at, and updated_at.
- Query Editor (Query 1):** A SQL query is run against the "conversations" table:

```
1 USE book_db;
2
3 SELECT id, user_id, title, created_at, updated_at
4 FROM conversations
5 ORDER BY id DESC
6 LIMIT 10;
```
- Result Grid:** The results of the query are displayed in a grid format. The columns are id, user_id, title, created_at, and updated_at. The data includes rows for "DB Proof", "Demo chat", "Test Chat", and "Test chat".
- Action Output:** A table showing the history of actions taken during the session, including the execution of the previous query and other database operations.
- Message Bar:** A message states: "Automatic context help is disabled. Use the toolbar to manually get help for the current caret position or to toggle automatic help."

Messages for one conversation :

Local instance 3306 - Warning - not supported

Administration Schemas Query 1 Context Help Snippets

SCHEMAS

book_db

- Tables
 - Books
 - conversations
 - Columns
 - id
 - user_id
 - title
 - created_at
 - updated_at
 - Indexes
 - Foreign Keys
 - Triggers
- messages
- Columns
 - id
 - conversation_id
 - role
 - content
 - created_at
- Indexes
- Foreign Keys
- Triggers
- Views
- Stored Procedures
- Functions
- sys

Limit to 1000 rows

1 USE book_db;

2

3 SELECT id, conversation_id, role, LEFT(content, 100) AS content_snippet, created_at

4 FROM messages

5 WHERE conversation_id = 1

6 ORDER BY id;

7

100% 26:5

Result Grid Filter Rows Search Export

id	conversation_id	role	content_snippet	created_at
1	1	user	Hello! Viral	2025-10-03 18:44:32
8	1	user	Give me a fun fact in one sentence.	2025-10-03 19:28:26
9	1	assistant	The shortest war in history was between Britain and Zanzibar on August 27, 1896, and lasted only 38	2025-10-03 19:28:40

Result 11 Read Only

Action Output

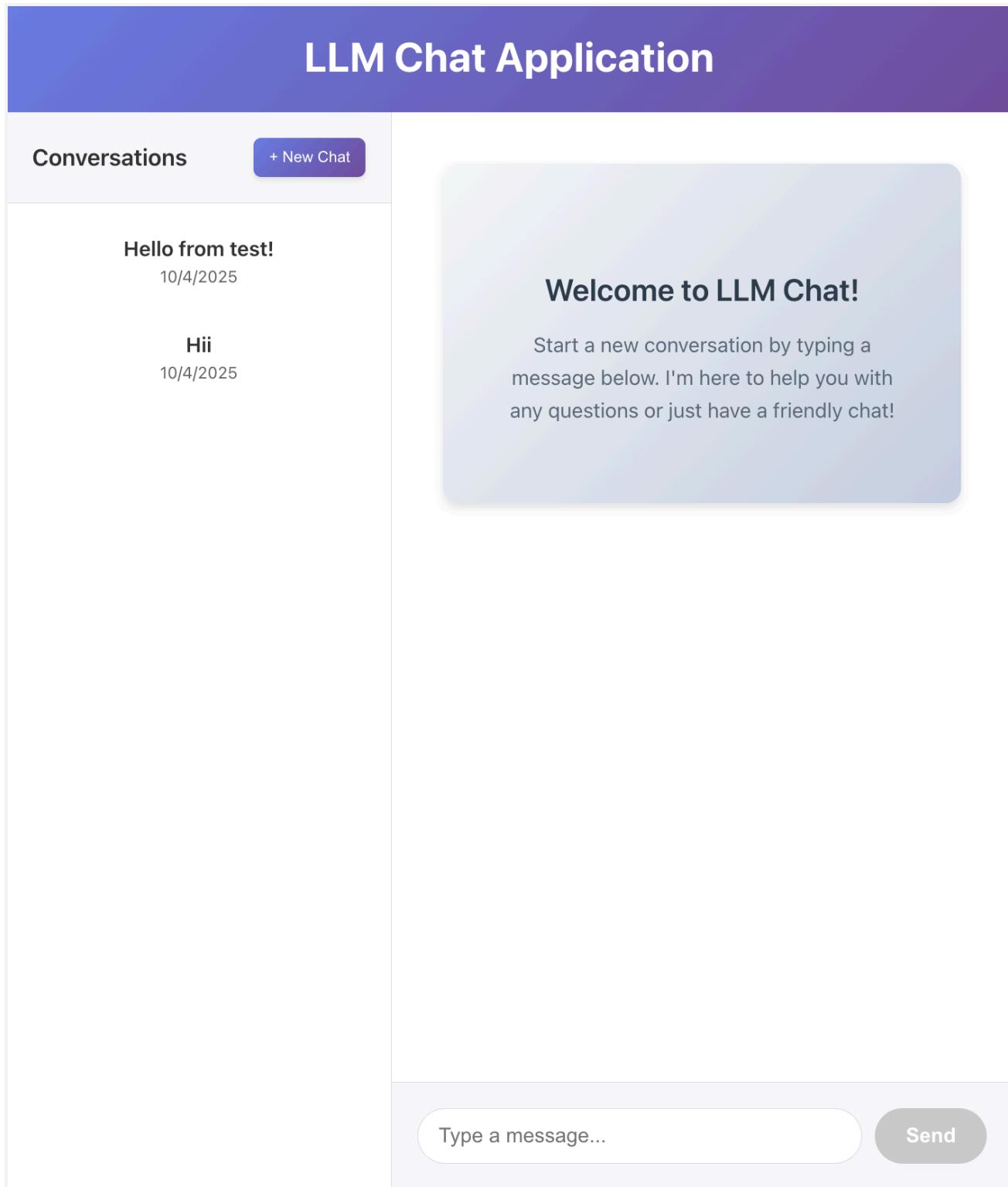
Time	Action	Response	Duration / Fetch Time...
19:54:00	SELECT id, user_id, title, created_at, updated_at FROM conversations ORDER BY id	7 row(s) returned	0.00000 sec / 0.00000...
7 19:53:37	SELECT id, conversation_id, role, LEFT(content, 100) AS content_snippet, created...	Error Code: 1064. You have an error in your SQL syntax...	0.00088 sec
8 19:54:19	SELECT id, conversation_id, role, LEFT(content, 100) AS content_snippet, created...	3 row(s) returned	0.00096 sec / 0.000...
9 19:54:29	SELECT id, conversation_id, role, LEFT(content, 100) AS content_snippet, created...	1 row(s) returned	0.00046 sec / 0.000...
10 19:54:32	SELECT id, conversation_id, role, LEFT(content, 100) AS content_snippet, created...	1 row(s) returned	0.00071 sec / 0.000...
11 19:54:35	SELECT id, conversation_id, role, LEFT(content, 100) AS content_snippet, created...	2 row(s) returned	0.00079 sec / 0.000...
12 19:54:38	SELECT id, conversation_id, role, LEFT(content, 100) AS content_snippet, created...	2 row(s) returned	0.00065 sec / 0.000...
13 19:54:46	SELECT id, conversation_id, role, LEFT(content, 100) AS content_snippet, created...	2 row(s) returned	0.00037 sec / 0.000...
14 19:54:49	SELECT id, conversation_id, role, LEFT(content, 100) AS content_snippet, created...	2 row(s) returned	0.00048 sec / 0.000...
15 19:54:54	SELECT id, conversation_id, role, LEFT(content, 100) AS content_snippet, created...	3 row(s) returned	0.00040 sec / 0.000...

Query Completed

Frontend — Redux Slice

- o **fetchConversations, fetchMessages, sendMessage thunks using Axios.**

Fetch Conversations:



The screenshot shows the Chrome DevTools Redux DevTools extension interface. The top navigation bar includes tabs for Elements, Console, Sources, **Redux**, Network, Performance, Memory, Application, Privacy and security, and several icons for export, refresh, and settings.

The left sidebar contains a list of actions with their timestamps:

- @@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]

The right panel displays a state diff between two snapshots. The top row has tabs for Action, State, Diff (which is selected), Trace, and Test. Below that, there are Tree and Raw tabs. The Diff view shows the following state structure:

```
{  
  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": "Hi!",  
        "created_at": "2025-10-04T23:06:22",  
        "updated_at": "2025-10-04T23:06:22"  
      }  
    ],  
    currentConversationId: null => 9,  
    status: "loading" => "succeeded"  
  },  
}
```

Fetch Messages:

LLM Chat Application

Conversations + New Chat

Hello from test!
10/4/2025

Hii
10/4/2025

User 11:52:43 PM
Hello from test!

Assistant 11:52:43 PM
It looks like you're testing something. What's on your mind? I'm here to chat if you need anything!

User 11:03:15 AM
Hey there!

Assistant 11:03:15 AM
It's nice to meet you. Is there something I can help you with or would you like to chat?

User 11:03:15 AM
Good Morning!

Assistant 11:03:15 AM
Good morning! Hope you're having a great start to the day! How can I assist you today?

User 11:03:15 AM
Hi Hello!

Assistant 11:03:15 AM
Hello! It's nice to meet you. Is there something I can help you with or would you like to chat?

User 11:03:15 AM
Hello!

Assistant 11:03:15 AM
Hello! It's nice to meet you. Is there something I can help you with, or would you like to chat?

User 11:03:15 AM
Hello! Hii

Assistant 11:03:15 AM
It looks like you're trying to say "Hii"! How can I assist you today? Do you have a question or need help with something specific?

Type a message... Send

The screenshot shows the Chrome DevTools Network tab with the "Redux" section selected. A "Diff" panel is open, comparing two states of a conversation log. The left pane shows the initial state with various API calls like "chat/fetchConversations/pending" and "chat/fetchMessages/pending". The right pane shows the final state with the actual conversation messages. The messages are highlighted in green boxes, indicating they were added or modified.

```
chat/fetchConversations/pending [11:24:25,63]
chat/fetchConversations/pending [+00:00:01]
chat/fetchConversations/fulfilled [+00:00:06]
chat/fetchMessages/pending [+00:00:00]
chat/fetchMessages/fulfilled [+00:00:01]
chat/fetchConversations/fulfilled [+00:00:06]
chat/startNewConversation [+00:01:28]
chat/setCurrentConversation [+00:13:70]
chat/fetchMessages/pending [+00:00:01]
chat/fetchMessages/fulfilled [+00:00:05]
chat/startNewConversation [+00:55:01]
chat/setCurrentConversation [+01:33:30]
chat/fetchMessages/pending [+00:00:00]
chat/fetchMessages/fulfilled [+00:00:05]
```

Diff

Tree Raw

```
[{"chat": {"messages": [{"id": 20, "role": "user", "content": "Hello from test!", "created_at": "2025-10-04T23:52:43"}, {"id": 21, "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"}, {"id": 22, "role": "user", "content": "Hey there!", "created_at": "2025-10-05T11:03:15"}, {"id": 23, "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"}, {"id": 24, "role": "user", "content": "Good morning!", "created_at": "2025-10-05T11:03:15"}, {"id": 25, "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"}, {"id": 26, "role": "user", "content": "Hi Hello!", "created_at": "2025-10-05T11:03:15"}, {"id": 27, "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"}, {"id": 28, "role": "user", "content": "Hello!", "created_at": "2025-10-05T11:03:15"}, {"id": 29, "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"}, {"id": 30, "role": "user", "content": "Hello! Hi!", "created_at": "2025-10-05T11:03:15"}, {"id": 31, "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" }]
```

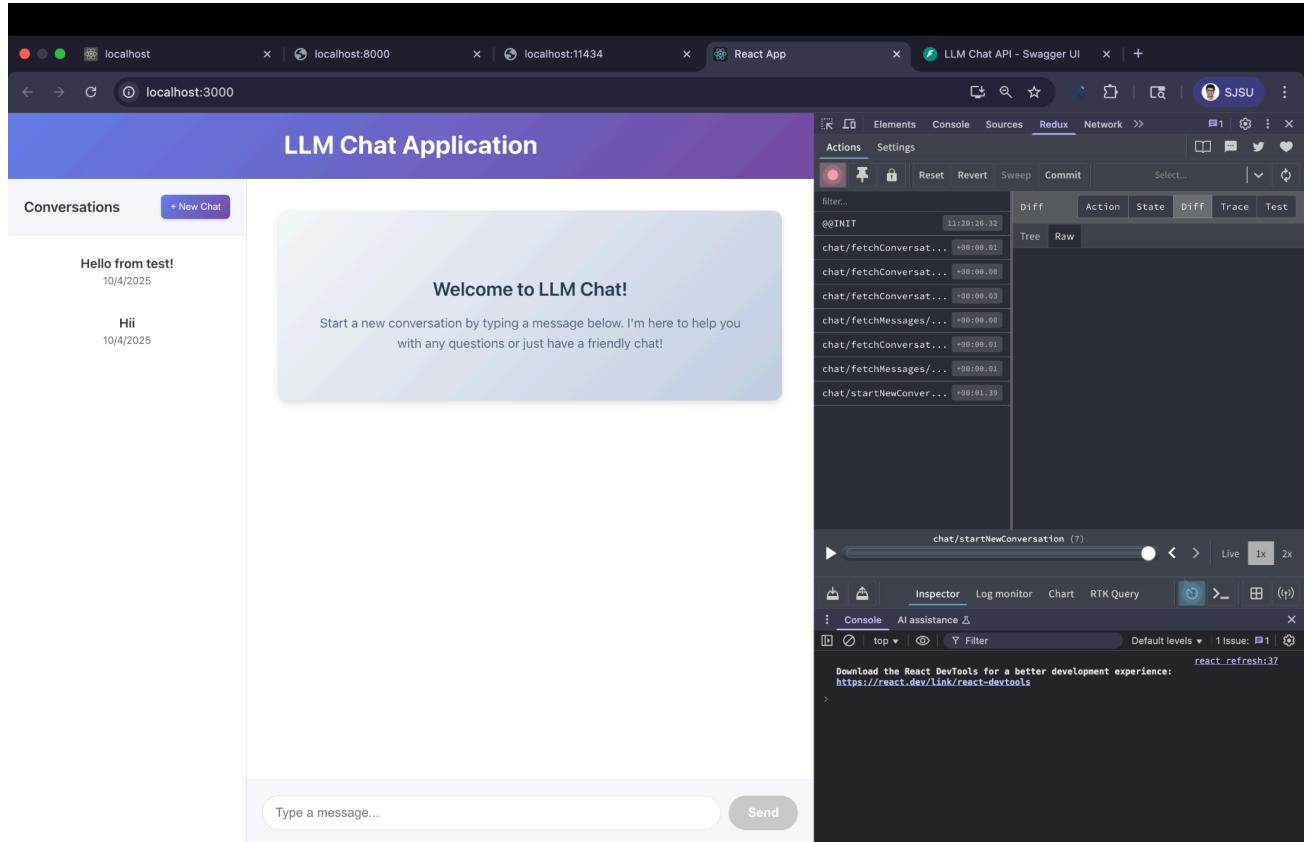
sendMessage:

The screenshot shows a web application titled "LLM Chat Application" running on localhost:3000. On the left, there's a sidebar with "Conversations" and a "+ New Chat" button. The main area displays a conversation history. A blue message bubble from the user contains "Hii" and the timestamp "10/5/2025". A grey message bubble from the "Assistant" at "4:55:55 PM" contains "How can I assist you today?". To the right, another blue message bubble from "User" at "4:55:53 PM" contains "Hii".

The screenshot shows the Redux DevTools interface with the "Diff" tab selected. The state tree shows a "chat" object with a "messages" array containing two items. The first item is a user message with id 1, role "user", content "Hello!", and created at "2025-10-05T12:22:46". The second item is an AI response with id 2, role "assistant", content "Hi there! How can I help you today?", and created at "2025-10-05T12:22:46".

```
{  
  chat: {  
    messages: [  
      0: {  
        "id": 1,  
        "role": "user",  
        "content": "Hello!",  
        "created_at": "2025-10-05T12:22:46"  
      },  
      1: {  
        "id": 2,  
        "role": "assistant",  
        "content": "Hi there! How can I help you today?",  
        "created_at": "2025-10-05T12:22:46"  
      }  
    ]  
  }  
}
```

LLM Chat UI:



Testing

cURL examples:

1) Start New Conversation:

```
● spartan@MLK-SCS-M7J3NJ9HTV backend % curl -X POST "http://localhost:8000/ai/chat" \
-H "Content-Type: application/json" \
-d '{
  "user_id": "user1",
  "message": "Hello! This is my first message.",
  "conversation_id": null,
  "title": "First Conversation"
}'
{"conversation_id":9,"reply":"Welcome to our conversation! It's great to have you here. Don't worry if you're feeling a bit uncertain - this is your first message, after all. How can I help or assist you today?"}%
```

2)Continue Existing Conversation:

```
● spartan@MLK-SCS-M7J3NJ9HTV backend % curl -X POST "http://localhost:8000/ai/chat" \
-H "Content-Type: application/json" \
-d '{
  "user_id": "user1",
  "message": "Can you tell me about artificial intelligence?",
  "conversation_id": 1,
  "title": "First Conversation"
}'

{"conversation_id":1,"reply":"Artificial Intelligence (AI) is a rapidly growing field that has been gaining significant attention recent years. I'd be happy to provide an overview of AI, its applications, and some of the latest developments.\n\n**What is Artificial Intelligence?**\nArtificial Intelligence refers to the development of computer systems that can perform tasks that typically require human intelligence, such as:\n\n1. **Learning**: The ability to learn from experience and improve performance over time.\n2. **Reasoning**: The ability to draw conclusions and make decisions based on available information.\n3. **Problem-solving**: The ability to identify and solve complex problems.\n4. **Perception**: The ability to interpret and understand sensory data, such as images or speech.\n\n**Types of Artificial Intelligence**\nThere are several types of AI, including:\n\n1. **Narrow or Weak AI**: Designed to perform a specific task, such as image recognition or natural language processing.\n2. **General or Strong AI**: A hypothetical type of AI that possesses the ability to understand, learn, and apply knowledge across a wide range of tasks.\n3. **Superintelligence**: An AI system that significantly surpasses human intelligence in various domains.\n\n**Applications of Artificial Intelligence**\nAI has numerous applications across various industries, including:\n\n1. **Healthcare**: Diagnosing diseases, predicting patient outcomes, and developing personalized treatment plans.\n2. **Customer Service**: Chatbots and virtual assistants that can provide 24/7 support to customers.\n3. **Finance**: Predictive analytics for stock trading, risk assessment, and credit scoring.\n4. **Transportation**: Self-driving cars, drones, and navigation systems.\n5. **Education**: Personalized learning platforms, grading systems, and intelligent tutoring.\n6. **Security**: Fraud detection, identity verification, and threat analysis.\n\n**Key Technologies**\nSome of the key technologies driving AI development include:\n\n1. **Machine Learning (ML)**: A subset of AI that involves training algorithms on large datasets to enable them to learn and make predictions or decisions.\n2. **Deep Learning (DL)**: A type of ML that uses neural networks with multiple layers to analyze complex data.\n3. **Natural Language Processing (NLP)**: The ability of computers to understand, interpret, and generate human language.\n4. **Computer Vision**: The ability of computers to interpret and understand visual data from images and videos.\n\n**Challenges and Ethics**\nAs AI continues to advance, it raises important questions about:\n\n1. **Bias and Fairness**: Ensuring that AI systems are fair, unbiased, and transparent in their decision-making processes.\n2. **Job Displacement**: The potential impact of AI on employment and the need for workers to develop skills relevant to an AI-driven economy.\n3. **Safety and Security**: Mitigating risks associated with AI, such as cyber attacks, data breaches, and autonomous vehicle accidents.\n\n**Recent Developments**\nSome notable recent developments in AI include:\n\n1. **Alexa and Google Assistant**: Virtual assistants that can perform a wide range of tasks, from setting reminders to controlling smart home devices.\n2. **Self-driving Cars**: Companies like Waymo and Tesla are making significant progress in developing autonomous vehicles.\n3. **DeepMind's AlphaGo**: A computer program that defeated a human world champion in the game of Go.\n\nThis is just a brief overview of AI, but I hope it gives you a starting point for exploring this fascinating field! Do you have any specific questions or topics you'd like to discuss further?"}
```

3)List All Conversations:

```
● spartan@MLK-SCS-M7J3NJ9HTV backend % curl -X GET "http://localhost:8000/ai/conversations?user_id=user1"
[{"id":9,"user_id":"user1","title":"Hello! This is my first message.", "created_at":"2025-10-05T15:30:35", "updated_at":"2025-10-05T15:30:35"}, {"id":8,"user_id":"user1","title":"What is machine learning?", "created_at":"2025-10-05T15:25:20", "updated_at":"2025-10-05T15:25:20"}, {"id":7,"user_id":"user1","title":"Hello! This is my first message.", "created_at":"2025-10-05T15:24:33", "updated_at":"2025-10-05T15:24:33"}, {"id":6,"user_id":"user1","title":"Test from curl", "created_at":"2025-10-05T15:23:07", "updated_at":"2025-10-05T15:23:07"}, {"id":5,"user_id":"user1","title":"Hello", "created_at":"2025-10-05T15:16:57", "updated_at":"2025-10-05T15:16:57"}, {"id":4,"user_id":"user1","title":"Test message from frontend", "created_at":"2025-10-05T15:13:57", "updated_at":"2025-10-05T15:13:57"}, {"id":3,"user_id":"user1","title":"Hello test", "created_at":"2025-10-05T15:13:06", "updated_at":"2025-10-05T15:13:06"}, {"id":1,"user_id":"user1","title":"Welcome to LLM Chat!", "created_at":"2025-10-05T15:12:57", "updated_at":"2025-10-05T15:12:57"}, {"id":2,"user_id":"user1","title":"Test Conversation", "created_at":"2025-10-05T15:12:57", "updated_at":"2025-10-05T15:12:57"}]
● spartan@MLK-SCS-M7J3NJ9HTV backend %
```

4)fetch messages:

```
● spartan@MLK-SCS-M7J3NJ9HTV backend % curl -X GET "http://localhost:8000/ai/messages/1"
{"conversation_id":1,"messages":[{"id":1,"role":"user","content":"Hello!","created_at":"2025-10-05T15:12:57"}, {"id":2,"role":"assistant","content":"Hi there! How can I help you today?","created_at":"2025-10-05T15:12:57"}, {"id":15,"role":"user","content":"Can you tell me about artificial intelligence?","created_at":"2025-10-05T15:24:38"}, {"id":16,"role":"assistant","content":"Artificial Intelligence (AI) is a vast and rapidly evolving field that has the potential to transform many aspects of our lives. I'll provide an overview, but feel free to ask specific questions or explore particular topics in more depth.\n\n**What is Artificial Intelligence?**\nArtificial Intelligence refers to the development of computer systems that can perform tasks that would typically require human intelligence, such as:\n1. Learning\n2. Problem-solving\n3. Reasoning\n4. Perception\n5. Understanding natural language\nThese capabilities enable AI systems to interact with humans and their environment in a more intelligent way.\n\n**Types of Artificial Intelligence:**\nThere are several subfields of AI, including:\n1. **Narrow or Weak AI:** Designed to perform a specific task, such as image recognition, speech recognition, or game playing.\n2. **General or Strong AI:** A hypothetical form of AI that would possess human-like intelligence and be able to learn, reason, and apply knowledge across multiple tasks.\n3. **Superintelligence:** AI that surpasses human intelligence in certain domains, enabling it to perform tasks that are currently beyond the capabilities of humans.\n\n**Applications of Artificial Intelligence:**\nAI is being applied in various fields, including:\n1. **Virtual Assistants:** Siri, Alexa, and Google Assistant use natural language processing (NLP) and machine learning (ML) to understand voice commands.\n2. **Image Recognition:** Facebook's facial recognition technology and self-driving cars rely on computer vision.\n3. **Healthcare:** AI is used in medical diagnosis, disease prediction, and personalized medicine.\n4. **Autonomous Vehicles:** Self-driving cars use a combination of sensors, mapping, and machine learning to navigate roads.\n5. **Customer Service:** Chatbots and virtual agents are being used to provide 24/7 customer support.\n\n**Technologies Behind Artificial Intelligence:**\nSome key technologies driving AI advancements include:\n1. **Machine Learning (ML):** A subset of AI that enables systems to learn from data without being explicitly programmed.\n2. **Deep Learning (DL):** A type of ML that uses neural networks with multiple layers to analyze complex data.\n3. **Natural Language Processing (NLP):** Enables computers to understand, interpret, and generate human language.\n4. **Computer Vision:** Allows machines to interpret visual data from images and videos.\n\n**Challenges and Concerns:**\nAs AI becomes increasingly prevalent, concerns have arisen about:\n1. **Bias and Fairness:** AI systems can inherit biases present in the training data, leading to unfair outcomes.\n2. **Job Displacement:** AI may automate jobs, potentially displacing human workers.\n3. **Security and Safety:** AI systems can be vulnerable to cyber attacks or errors that lead to accidents.\n\n**The Future of Artificial Intelligence:**\nResearchers continue to advance AI capabilities, we can expect:\n1. **Increased Adoption:** AI will become more widespread in various industries and aspects of life.\n2. **Improved Efficiency:** AI will automate routine tasks and free humans for more creative and complex work.\n3. **Advancements in Healthcare:** AI-driven medical research and personalized medicine may lead to breakthroughs.\n\nThat's a brief overview of Artificial Intelligence! Is there a specific aspect you'd like me to expand on or discuss further?", "created_at": "2025-10-05T15:25:19"}, {"id":21, "role": "user", "content": "Can you tell me about artificial intelligence?", "created_at": "2025-10-05T15:32:00"}, {"id":22, "role": "assistant", "content": "Artificial Intelligence (AI) is a rapidly growing field that has been gaining significant attention in recent years. I'd be happy to provide an overview of AI, its applications, and some of the latest developments.\n\n**What is Artificial Intelligence?**\nArtificial Intelligence refers to the development of computer systems that can perform tasks that typically require human intelligence, such as:\n1. **Learning:** The ability to learn from experience and improve performance over time.\n2. **Reasoning:** The ability to draw conclusions and make decisions based on available information.\n3. **Problem-solving:** The ability to identify and solve complex problems.\n4. **Perception:** The ability to interpret and understand sensory data, such as images and speech.\n\n**Types of Artificial Intelligence:**\nThere are several types of AI, including:\n1. **Narrow or Weak AI:** Designed to perform a specific task, such as image recognition or natural language processing.\n2. **General or Strong AI:** A hypothetical type of AI that possesses the ability to understand, learn, and apply knowledge across a wide range of tasks.\n3. **Superintelligence:** An AI system that significantly surpasses human intelligence in various domains.\n\n**Applications of Artificial Intelligence:**\nAI has numerous applications across various industries, including:\n1. **Healthcare:** Diagnosing diseases, predicting patient outcomes, and developing personalized treatment plans.\n2. **Customer Service:** Chatbots and virtual assistants that can provide 24/7 support to customers.\n3. **Finance:** Predictive analytics for stock trading, risk assessment, and credit scoring.\n4. **Transportation:** Self-driving cars, drones, and navigation systems.\n5. **Education:** Personalized learning platforms, grading systems, and intelligent tutoring.\n6. **Security:** Fraud detection, identity verification, and threat analysis.\n\n**Key Technologies:**\nSome of the key technologies driving AI development include:\n1. **Machine Learning (ML):** A subset of AI that involves training algorithms on large datasets to enable them to learn and make predictions or decisions.\n2. **Deep Learning (DL):** A type of ML that uses neural networks with multiple layers to analyze complex data.\n3. **Natural Language Processing (NLP):** The ability of computers to understand, interpret, and generate human language.\n4. **Computer Vision:** The ability of computers to interpret and understand visual data from images and videos.\n\n**Challenges and Ethics:**\nAs AI continues to advance, it raises important questions about:\n1. **Bias and Fairness:** Ensuring that AI systems are fair, unbiased, and transparent in their decision-making processes.\n2. **Job Displacement:** The potential impact of AI on employment and the need for workers to develop skills relevant to an AI-driven economy.\n3. **Security and Safety:** Mitigating risks associated with AI, such as cyber attacks, data breaches, and autonomous vehicle accidents.\n\n**Recent Developments:**\nSome notable recent developments in AI include:\n1. **Alexa and Google Assistant:** Virtual assistants that can perform a wide range of tasks, from setting reminders to controlling smart home devices.\n2. **Self-driving Cars:** Companies like Waymo and Tesla are making significant progress in developing autonomous vehicles.\n3. **DeepMind's AlphaGo:** A computer program that defeated a human world champion in the game of Go.\n\nThis is just a brief overview of AI, but I hope it gives you a good starting point for exploring this fascinating field! Do you have any specific questions or topics you'd like to discuss further?", "created_at": "2025-10-05T15:32:45"}]}% spartan@MLK-SCS-M7J3NJ9HTV backend %
```

CODE:

The screenshot shows a file explorer interface with the following project structure:

- 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 X ChatApp.tsx U tsconfig.json library_router.py database.py main.py .../Assignment > Assignment 5 > llm_chat > backend > router.py > chat
1  from fastapi import APIRouter, Depends, HTTPException
2  from sqlalchemy.orm import Session
3  from typing import List
4  from datetime import datetime
5  import models
6  import schemas
7  from database import get_db
8  import httpx
9
10 router = APIRouter(prefix="/ai", tags=["ai"])
11
12 @router.get("/test")
13 def test_endpoint():
14     try:
15         from database import get_db
16         from sqlalchemy.orm import Session
17         db = next(get_db())
18
19         conversations = db.query(models.Conversation).all()
20
21         return {
22             "status": "success",
23             "message": "Database connection working",
24             "conversations_count": len(conversations)
25         }
26     except Exception as e:
27         return {
28             "status": "error",
29             "message": str(e),
30             "error_type": type(e).__name__
31         }
32
33 async def call_ollama(message: str, model: str = "llama3.1:latest") -> str:
34     async with httpx.AsyncClient(timeout=httpx.Timeout(120.0)) as client:
35         try:
36             response = await client.post(
37                 "http://localhost:11434/api/generate",
38                 json={
39                     "model": model,
40                     "prompt": message,
41                     "stream": False
42                 }
43         )
44         response.raise_for_status()
45         data = response.json()
46         return data.get("response", "")
47     except httpx.RequestError as e:
48         raise HTTPException(status_code=503, detail=f"Ollama service unavailable: {str(e)}")
49     except httpx.HTTPStatusError as e:
50         raise HTTPException(status_code=e.response.status_code, detail=f"Ollama API error: {str(e)}")
51
52 @router.post("/chat", response_model=schemas.ChatOut)
53 async def chat(input: schemas.ChatIn, db: Session = Depends(get_db)):
54     try:
55         if input.conversation_id:
56             conv = db.query(models.Conversation).filter_by(id=input.conversation_id).first()
57             if not conv:
58                 raise HTTPException(status_code=404, detail="Conversation not found")
59         else:
60             now = datetime.now()
61             conv = models.Conversation(
62                 user_id=input.user_id,
63                 title=input.title or input.message[:40] + "..." if len(input.message) > 40 else input.message,
64                 created_at=now,
65                 updated_at=now
66             )
67             db.add(conv)
68             db.commit()
69             db.refresh(conv)
70
71             now = datetime.now()
72             user_msg = models.Message(
73                 conversation_id=conv.id,
74                 role=models.RoleEnum.user,
75                 content=input.message,
76                 created_at=now
77             )
78             db.add(user_msg)
79             db.commit()
80
```

```
❸ router.py .../backend ❹ ChatApp.tsx U ❺ tsconfig.json ❻ library_router.py ❼ database.py ❼ main.py .../Assignment 5 ❽
Assignment > Assignment 5 > llm_chat > backend > ❶ router.py > ❷ chat
53     async def chat(input: schemas.ChatIn, db: Session = Depends(get_db)):
54         )
55         db.add(user_msg)
56         db.commit()
57
58         try:
59             reply = await call_ollama(input.message)
60         except HTTPException:
61             raise
62         except Exception as e:
63             raise HTTPException(status_code=500, detail=f"Error calling Ollama: {str(e)}")
64
65         now = datetime.now()
66         assistant_msg = models.Message(
67             conversation_id=conv.id,
68             role=models.RoleEnum.assistant,
69             content=reply,
70             created_at=now
71         )
72         db.add(assistant_msg)
73         db.commit()
74
75         return schemas.ChatOut(conversation_id=conv.id, reply=reply)
76     except Exception as e:
77         print(f"Error in chat endpoint: {e}")
78         import traceback
79         traceback.print_exc()
80         raise HTTPException(status_code=500, detail=f"Internal server error: {str(e)}")
81
82 @router.get("/conversations", response_model=List[schemas.ConversationOut])
83 def get_conversations(user_id: str, db: Session = Depends(get_db)):
84     conversations = db.query(models.Conversation).filter_by(user_id=user_id).all()
85
86     valid_conversations = []
87     for conv in conversations:
88         try:
89             if conv.created_at is None or conv.updated_at is None:
90                 continue
91             if str(conv.created_at) == 'None' or str(conv.updated_at) == 'None':
92                 continue
93             if str(conv.created_at) == '' or str(conv.updated_at) == '':
94                 continue
95             if not isinstance(conv.created_at, datetime) or not isinstance(conv.updated_at, datetime):
96                 continue
97             valid_conversations.append(conv)
98         except Exception as e:
99             continue
100
101     valid_conversations.sort(key=lambda x: x.updated_at, reverse=True)
102
103     return valid_conversations
104
105 @router.get("/messages/{conversation_id}", response_model=schemas.MessagesOut)
106 def get_messages(conversation_id: int, db: Session = Depends(get_db)):
107     """Get all messages in a conversation"""
108     conv = db.query(models.Conversation).filter_by(id=conversation_id).first()
109     if not conv:
110         raise HTTPException(status_code=404, detail="Conversation not found")
111
112     messages = db.query(models.Message).filter_by(conversation_id=conversation_id).order_by(models.Message.created_at.asc()).all()
113
114     valid_messages = []
115     for m in messages:
116         try:
117             if m.created_at is None:
118                 continue
119             if str(m.created_at) == 'None':
120                 continue
121             if str(m.created_at) == '':
122                 continue
123             valid_messages.append(m)
124         except Exception as e:
125             continue
126
127     return schemas.MessagesOut(
128         conversation_id=conversation_id,
129         messages=[
```

```
136
137     valid_messages = []
138     for m in messages:
139         try:
140             if m.created_at is None:
141                 continue
142             if str(m.created_at) == 'None':
143                 continue
144             if str(m.created_at) == '':
145                 continue
146             valid_messages.append(m)
147         except Exception as e:
148             continue
149
150     return schemas.MessagesOut(
151         conversation_id=conversation_id,
152         messages=[
153             schemas.ChatMessageOut(
154                 id=m.id,
155                 role=m.role.value,
156                 content=m.content,
157                 created_at=m.created_at
158             ) for m in valid_messages
159         ]
160     )
161
```

```
models.py .../Assignment 5      schemas.py .../backend X    router.py .../Assignment 5
Assignment > Assignment 5 > llm_chat > backend > schemas.py > ChatIn
1  from pydantic import BaseModel, field_validator
2  from datetime import datetime
3  from typing import Optional, List
4
5  class ChatIn(BaseModel):
6      user_id: str
7      message: str
8      conversation_id: Optional[int] = None
9      title: Optional[str] = None
10
11 class ChatOut(BaseModel):
12     conversation_id: int
13     reply: str
14
15 class ChatMessageOut(BaseModel):
16     id: int
17     role: str
18     content: str
19     created_at: datetime
20
21     @field_validator('created_at', mode='before')
22     @classmethod
23     def validate_datetime(cls, v):
24         if v is None or v == '' or v == 'None':
25             return datetime.now()
26         return v
27
28 class MessagesOut(BaseModel):
29     conversation_id: int
30     messages: List[ChatMessageOut]
31
32 class ConversationOut(BaseModel):
33     id: int
34     user_id: str
35     title: Optional[str]
36     created_at: datetime
37     updated_at: datetime
38
39     @field_validator('created_at', 'updated_at', mode='before')
40     @classmethod
41     def validate_datetime(cls, v):
42         if v is None or v == '' or v == 'None':
43             return datetime.now()
44         return v
45
46     class Config:
47         from_attributes = True
48
```

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

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

```
1  from sqlalchemy import create_engine
2  from sqlalchemy.orm import sessionmaker
3  import os
4  from dotenv import load_dotenv
5
6  load_dotenv()
7
8  DB_HOST = os.getenv("DB_HOST", "localhost")
9  DB_PORT = os.getenv("DB_PORT", "3306")
10 DB_USER = os.getenv("DB_USER", "root")
11 DB_PASSWORD = os.getenv("DB_PASSWORD", "pass1234")
12 DB_NAME = os.getenv("DB_NAME", "book_db")
13
14 MYSQL_URL = f"mysql+pymysql://{{DB_USER}}:{{DB_PASSWORD}}@{{DB_HOST}}:{{DB_PORT}}/{{DB_NAME}}"
15
16 engine = create_engine(SQLALCHEMY_DATABASE_URL)
17 SessionLocal = sessionmaker(autocommit=False, autoflush=False, bind=engine)
18
19 def get_db():
20     db = SessionLocal()
21     try:
22         yield db
23     finally:
24         db.close()
25
```

The screenshot shows a code editor interface with multiple tabs at the top. The active tab is 'models.py' located in the 'backend' directory. The code itself is a Python file defining two database models: 'Conversation' and 'Message'. The 'Conversation' model has columns for id, user_id, title, created_at, and updated_at. It also has a relationship named 'messages' back_populates='conversation' with the 'Message' model. The 'Message' model has columns for id, conversation_id, role, content, and created_at. It has a relationship named 'conversation' back_populates='messages' with the 'Conversation' model. The code uses SQLAlchemy's declarative base and foreign key relationships.

```
Assignment > Assignment 5 > llm_chat > backend > models.py > ...
1  from sqlalchemy import Column, Integer, String, ForeignKey, Enum, Text, DateTime, func
2  from sqlalchemy.orm import relationship, declarative_base
3  import enum
4
5  Base = declarative_base()
6
7  class RoleEnum(enum.Enum):
8      user = "user"
9      assistant = "assistant"
10     system = "system"
11
12 class Conversation(Base):
13     __tablename__ = "conversations"
14
15     id = Column(Integer, primary_key=True, index=True)
16     user_id = Column(String(64), nullable=False)
17     title = Column(String(128), nullable=True)
18     created_at = Column(DateTime(timezone=True), server_default=func.now())
19     updated_at = Column(DateTime(timezone=True), server_default=func.now(), onupdate=func.now())
20
21     messages = relationship("Message", back_populates="conversation", cascade="all, delete-orphan")
22
23 class Message(Base):
24     __tablename__ = "messages"
25
26     id = Column(Integer, primary_key=True, index=True)
27     conversation_id = Column(Integer, ForeignKey("conversations.id"), nullable=False)
28     role = Column(Enum(RoleEnum), nullable=False)
29     content = Column(Text, nullable=False)
30     created_at = Column(DateTime(timezone=True), server_default=func.now())
31
32     conversation = relationship("Conversation", back_populates="messages")
```

```
❶ main.py .../backend X ❷ models.py .../backend ❸ router.py .../backend
Assignment > Assignment 5 > llm_chat > backend > ❹ main.py > ❺ health_check
1   from fastapi import FastAPI
2   from fastapi.middleware.cors import CORSMiddleware
3   from database import engine
4   from models import Base
5   import router
6
7   app = FastAPI(
8       title="LLM Chat API",
9       version="1.0.0"
10      )
11
12  app.add_middleware(
13      CORSMiddleware,
14      allow_origins=["*"],
15      allow_credentials=True,
16      allow_methods=["*"],
17      allow_headers=["*"],
18      )
19
20 app.include_router(router.router)
21
22 @app.on_event("startup")
23 def on_startup():
24     Base.metadata.create_all(bind=engine)
25     print("Database tables created successfully!")
26     print("LLM Chat API is ready!")
27
28 @app.get("/")
29 def read_root():
30     return {"message": "LLM Chat API is running!", "docs": "/docs"}
31
32 @app.get("/health")
33 def health_check():
34     return {"status": "healthy", "service": "LLM Chat API"}
35
36 if __name__ == "__main__":
37     import uvicorn
38     uvicorn.run(app, host="0.0.0.0", port=8000)
```

```
Assignment > Assignment 5 > llm_chat > frontend > src > components > ChatApp.tsx > handleSend
1 import React, { useEffect, useState } from 'react';
2 import { useDispatch, useSelector } from 'react-redux';
3 import { fetchConversations, fetchMessages, sendMessage, setCurrentConversation, startNewConversation } from '../store/chatSlice';
4 import { RootState, AppDispatch } from '../store/store';
5 import './ChatApp.css';
6
7 const ChatApp: React.FC = () => {
8   const dispatch = useDispatch();
9   const { conversations, messages, currentConversationId, isNewConversation, userId, status, error } = useSelector((state: RootState) => state.chat);
10  const [text, setText] = useState('');
11
12  useEffect(() => {
13    dispatch(fetchConversations(userId));
14  }, [dispatch, userId]);
15
16  useEffect(() => {
17    if (currentConversationId && !isNewConversation) {
18      dispatch(fetchMessages(currentConversationId));
19    }
20  }, [dispatch, currentConversationId, isNewConversation]);
21
22  useEffect(() => {
23    if (currentConversationId) {
24      dispatch(fetchConversations(userId));
25    }
26  }, [dispatch, currentConversationId, userId]);
27
28
29  const handleSend = async (e: React.FormEvent) => {
30    e.preventDefault();
31    if (!text.trim()) return;
32
33    console.log('ChatApp: Starting send message process...');
34    console.log(`ChatApp: Current state:`, {
35      userId,
36      text,
37      currentConversationId,
38      isNewConversation
39    });
40
41    const messageText = text;
42    setText('');
43
44    try {
45      const result = await dispatch(sendMessage({
46        userId,
47        message: messageText,
48        conversationId: currentConversationId || undefined,
49        title: messageText.slice(0, 40)
50      }));
51
52      console.log('ChatApp: Send message result:', result);
53
54      if (sendMessage.fulfilled.match(result)) {
55        console.log('ChatApp: Message sent successfully, fetching messages...');
56        if (result.payload.conversation_id) {
57          await dispatch(fetchMessages(result.payload.conversation_id));
58        }
59        await dispatch(fetchConversations(userId));
60      } else if (sendMessage.rejected.match(result)) {
61        console.log('ChatApp: Message send rejected:', result.payload);
62      } else {
63        console.log('ChatApp: Message send in unknown state:', result);
64      }
65    } catch (error) {
66      console.error('ChatApp: Error in handleSend:', error);
67    }
68  };
69
70  const handleConversationSelect = (conversationId: number) => {
71    dispatch(setCurrentConversation(conversationId));
72  };
73
74  const handleNewConversation = () => {
75    dispatch(startNewConversation());
76    setText('');
77  };

```

```
Assignment > Assignment 5 > llm_chat > frontend > src > components > ChatApp.tsx > ChatApp > handleSend
  7  const ChatApp: React.FC = () => {
  8    const handleNewConversation = () => {
  9      dispatch(startNewConversation());
 10      setText('');
 11    };
 12
 13    const testConnection = async () => {
 14      try {
 15        console.log('Testing backend connection...');
 16        const response = await fetch('http://localhost:8000/health');
 17        const data = await response.json();
 18        console.log('Backend health check:', data);
 19        alert(`Backend is running! Status: ${data.status}`);
 20      } catch (error) {
 21        console.error('Backend connection failed:', error);
 22        const errorMessage = error instanceof Error ? error.message : 'Unknown error';
 23        alert(`Backend connection failed: ${errorMessage}`);
 24      }
 25    };
 26
 27    return (
 28      <div className="chat-app">
 29        <div className="chat-header">
 30          <h1>LLM Chat Application</h1>
 31          <button onClick={testConnection} style={{marginLeft: '20px', padding: '5px 10px'}}>
 32            Test Connection
 33          </button>
 34        </div>
 35
 36        <div className="chat-container">
 37          <div className="conversations-sidebar">
 38            <div className="conversations-header">
 39              <h3>Conversations</h3>
 40              <button
 41                onClick={handleNewConversation}
 42                className="new-conversation-btn"
 43                title="Start a new conversation"
 44              >
 45                + New Chat
 46              </button>
 47            </div>
 48            <div className="conversations-list">
 49              {conversations.map(conv => (
 50                <div
 51                  key={conv.id}
 52                  className={`conversation-item ${currentConversationId === conv.id ? 'active' : ''}`}
 53                  onClick={() => handleConversationSelect(conv.id)}
 54                >
 55                  <div className="conversation-title">{conv.title || 'New Chat'}</div>
 56                  <div className="conversation-date">
 57                    {new Date(conv.created_at).toLocaleDateString()}
 58                  </div>
 59                </div>
 60              ))}
 61              {conversations.length === 0 && (
 62                <div className="no-conversations">No conversations yet</div>
 63              )}
 64            </div>
 65          </div>
 66
 67          <div className="chat-main">
 68            <div className="messages-container">
 69              {messages.map(message => (
 70                <div key={message.id} className={`message ${message.role}`}>
 71                  <div className="message-header">
 72                    <span className="message-role">{message.role}</span>
 73                    <span className="message-time">
 74                      {new Date(message.created_at).toLocaleTimeString()}
 75                    </span>
 76                  </div>
 77                  <div className="message-content">{message.content}</div>
 78                </div>
 79              ))}
 80              {messages.length === 0 && isNewConversation && (
 81                <div>New message</div>
 82              )}
 83            </div>
 84          </div>
 85        </div>
 86      </div>
 87    );
 88  
```

```
Assignment > Assignment 5 > llm_chat > frontend > src > components > ChatApp.tsx > ChatApp > handleSend
  7  const ChatApp: React.FC = () => {
 137    <div>
 138      <div>
 139        <div>
 140          <div>
 141            <div>
 142              <div>
 143                <div>
 144                  <div>
 145                    <div>
 146                      <div>
 147                        <div>
 148                          <div>
 149                            <div>
 150                              <div>
 151                                <div>
 152                                  <div>
 153                                    <div>
 154                                      <div>
 155                                        <div>
 156                                          <div>
 157                                            <div>
 158                                              <div>
 159                                                <div>
 160                                                  <div>
 161                                                    <div>
 162                                                      <div>
 163                                                        <div>
 164                                                          <div>
 165                                                            <div>
 166                                                              <div>
 167                                                                <div>
 168                                                                  <div>
 169                                                                    <div>
 170                                                                      <div>
 171                                                                        <div>
 172                                                                          <div>
 173                                                                            <div>
 174                                                                              <div>
 175                                                                                <div>
 176                                                                                  <div>
 177                                                                                    <div>
 178                                                                 <div>
 179
 180          <div>
 181            <div>
 182              <div>
 183                <div>
 184                  <div>
 185                    <div>
 186                      <div>
 187                        <div>
 188                          <div>
 189                            <div>
 190
 191  export default ChatApp;
```

```
ts chatSlice.ts U X  models.py .../backend  ChatApp.tsx  router.py .../backend  library_router.py  database.py  main.p
Assignment > Assignment 5 > llm_chat > frontend > src > store > ts chatSlice.ts > [!] fetchConversations > [!] createAsyncThunk('chat/fetchConversations') callback
  1 import { createSlice, createAsyncThunk } from '@reduxjs/toolkit';
  2 import axios from 'axios';
  3
  4 const API_BASE_URL = 'http://localhost:8000';
  5
  6 const chatApi = axios.create({
  7   baseURL: API_BASE_URL,
  8   timeout: 30000,
  9   headers: {
 10     'Content-Type': 'application/json',
 11   }
 12 });
 13
 14 chatApi.interceptors.request.use(
 15   (config) => {
 16     console.log('API Request:', config.method?.toUpperCase(), config.url, config.data);
 17     return config;
 18   },
 19   (error) => {
 20     console.error('API Request Error:', error);
 21     return Promise.reject(error);
 22   }
 23 );
 24
 25 chatApi.interceptors.response.use(
 26   (response) => {
 27     console.log('API Response:', response.status, response.data);
 28     return response;
 29   },
 30   (error) => {
 31     console.error('API Response Error:', error.response?.status, error.response?.data);
 32     return Promise.reject(error);
 33   }
 34 );
 35
 36 export const fetchConversations = createAsyncThunk(
 37   'chat/fetchConversations',
 38   async (userId: string, thunkAPI) => {
 39     try {
 40       console.log('Fetching conversations for user:', userId);
 41       const { data } = await chatApi.get('/ai/conversations', {
 42         params: { user_id: userId }
 43       });
 44       console.log('Conversations fetched successfully:', data);
 45       return data;
 46     } catch (err: any) {
 47       console.error('Fetch conversations error:', err);
 48       console.error('Error details:', {
 49         message: err.message,
 50         response: err.response?.data,
 51         status: err.response?.status,
 52         statusText: err.response?.statusText
 53       });
 54       return thunkAPI.rejectWithValue(err.response?.data ?? { detail: 'Failed to load conversations' });
 55     }
 56   }
 57 );
 58
 59 export const fetchMessages = createAsyncThunk(
 60   'chat/fetchMessages',
 61   async (conversationId: number, thunkAPI) => {
 62     try {
 63       const { data } = await chatApi.get('/ai/messages/${conversationId}');
 64       return data;
 65     } catch (err: any) {
 66       console.error('Fetch messages error:', err);
 67       return thunkAPI.rejectWithValue(err.response?.data ?? { detail: 'Failed to load messages' });
 68     }
 69   }
 70 );
 71
 72 export const sendMessage = createAsyncThunk(
 73   'chat/sendMessage',
 74   async ({ userId, message, conversationId, title }: {
 75     userId: string;
 76     message: string;
 77     conversationId?: number;
 78   }) =>
```

```
ts chatSlice.ts U X  models.py .../backend ChatApp.tsx U  router.py .../backend library_router.py  database.py

Assignment > Assignment 5 > llm_chat > frontend > src > store > ts chatSlice.ts > (fetchConversations) > (createAsyncThunk('chat/fetchConversations')
72  export const sendMessage = createAsyncThunk(
73    async ({ userId, message, conversationId, title }: {
74      userId: string;
75      conversationId?: number;
76      title?: string;
77    }, thunkAPI) => {
78      console.log('Redux: Sending message to API:', { userId, message, conversationId, title });
79
80      const payload = {
81        user_id: userId,
82        message,
83        conversation_id: conversationId ?? null,
84        title: title ?? message.slice(0, 40),
85      };
86
87      console.log('Redux: Payload prepared:', payload);
88
89      try {
90        const response = await chatApi.post('/ai/chat', payload);
91        console.log('Redux: API response received:', response.data);
92
93        console.log('Redux: Returning data:', {
94          conversation_id: response.data.conversation_id,
95          reply: response.data.reply
96        });
97
98        return {
99          conversation_id: response.data.conversation_id,
100         reply: response.data.reply,
101         success: true
102       };
103     } catch (error) {
104       console.error('Redux: sendMessage error caught:', error);
105       return {
106         conversation_id: conversationId || 0,
107         reply: 'Error: Failed to send message',
108         success: false,
109         error: error instanceof Error ? error.message : 'Unknown error'
110       };
111     }
112   };
113 );
114
115 );
116
117 const chatSlice = createSlice({
118   name: 'chat',
119   initialState: {
120     userId: 'user1',
121     conversations: [] as any[],
122     messages: [] as any[],
123     currentConversationId: null as number | null,
124     isNewConversation: false,
125     status: 'idle' as 'idle' | 'loading' | 'succeeded' | 'failed',
126     error: null as any,
127   },
128   reducers: {
129     setCurrentConversation: (state, action) => {
130       state.currentConversationId = action.payload;
131       state.isNewConversation = action.payload === null;
132     },
133     startNewConversation: (state) => {
134       state.currentConversationId = null;
135       state.isNewConversation = true;
136       state.messages = [];
137     },
138     clearError: (state) => {
139       state.error = null;
140     },
141   },
142   extraReducers: (builder) => {
143     builder
144       .addCase(fetchConversations.pending, (state) => {
145         state.status = 'loading';
146         state.error = null;
147       })
148       .addCase(fetchConversations.fulfilled, (state, action) => {
149         state.status = 'succeeded';
150         state.conversations = action.payload;
151       })
152       .addCase(fetchConversations.rejected, (state, action) => {
153         state.status = 'failed';
154         state.error = action.error.message;
155       })
156   },
157 });
158
159 
```

```
ts chatSlice.ts U ✘ | models.py ...backend | ChatApp.tsx U | router.py ...backend | library_router.py | database.py | main.js
Assignment > Assignment 5 > llm_chat > frontend > src > store > ts chatSlice.ts > [!] fetchConversations > [!] createAsyncThunk('chat/fetchConversations') callback
117 const chatSlice = createSlice({
118   ,
119   extraReducers: (builder) => {
120     builder
121       .addCase(fetchConversations.pending, (state) => {
122         state.status = 'loading';
123         state.error = null;
124       })
125       .addCase(fetchConversations.fulfilled, (state, action) => {
126         state.status = 'succeeded';
127         state.conversations = action.payload;
128         if (!state.isNewConversation && state.currentConversationId === null && action.payload.length > 0) {
129           state.currentConversationId = action.payload[0].id;
130         }
131       })
132       .addCase(fetchConversations.rejected, (state, action) => {
133         state.status = 'failed';
134         state.error = action.payload || action.error;
135       })
136       .addCase(fetchMessages.pending, (state) => {
137         state.status = 'loading';
138         state.error = null;
139       })
140       .addCase(fetchMessages.fulfilled, (state, action) => {
141         state.status = 'succeeded';
142         state.messages = action.payload.messages || [];
143         state.currentConversationId = action.payload.conversation_id ?? state.currentConversationId;
144         state.isNewConversation = false;
145       })
146       .addCase(fetchMessages.rejected, (state, action) => {
147         state.status = 'failed';
148         state.error = action.payload || action.error;
149         if (state.isNewConversation) {
150           state.status = 'succeeded';
151           state.error = null;
152         }
153       })
154       .addCase(sendMessage.pending, (state) => {
155         console.log('Redux: sendMessage.pending');
156         state.status = 'loading';
157         state.error = null;
158       })
159       .addCase(sendMessage.fulfilled, (state, action) => {
160         console.log('Redux: sendMessage.fulfilled with payload:', action.payload);
161         state.status = 'succeeded';
162         state.currentConversationId = action.payload.conversation_id;
163         state.isNewConversation = false;
164
165         if (action.payload.success) {
166           state.error = null;
167           console.log('Redux: Message sent successfully, conversation_id:', action.payload.conversation_id);
168         } else {
169           state.error = { detail: action.payload.error || 'Failed to send message' };
170           console.log('Redux: Message send failed:', action.payload.error);
171         }
172       })
173       .addCase(sendMessage.rejected, (state, action) => {
174         console.log('Redux: sendMessage.rejected with payload:', action.payload);
175         state.status = 'failed';
176         state.error = action.payload || action.error;
177       });
178   },
179 });
180
181 export const { setCurrentConversation, startNewConversation, clearError } = chatSlice.actions;
182 export default chatSlice.reducer;
```

WORKING CHAT:

LLM Chat Application

Conversations + New Chat

Hello from test!
10/4/2025

Hii
10/4/2025

Assistant 11:52:43 PM
It looks like you're testing something. What's on your mind? I'm here to chat if you need anything!

User 11:52:43 PM
Hello from test!

Assistant 11:03:15 AM
It's nice to meet you. Is there something I can help you with or would you like to chat?

User 11:03:15 AM
Hey there!

Assistant 11:03:15 AM
Good morning! Hope you're having a great start to the day! How can I assist you today?

User 11:03:15 AM
Good Morning!

Assistant 11:03:15 AM
Hello! It's nice to meet you. Is there something I can help you with or would you like to chat?

User 11:03:15 AM
Hi Hello!

Assistant 11:03:15 AM
Hello! It's nice to meet you. Is there something I can help you with, or would you like to chat?

User 11:03:15 AM
Hello!

Assistant 11:03:15 AM
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?

User 11:03:15 AM
Hello! Hii

Type a message... Send