Great 👍

Now I'll explain this **React** `handleSubmit` **code** in the **same way** I explained:

> **FastAPI → JSON → Schema → Object → DB**

We'll do:

- Line-by-line explanation
- JSON → JS object → Backend schema mapping
- Graphical flow

🔹 Your Code

```javascript
const handleSubmit = async (e) => {
  e.preventDefault();

  const response = await fetch("http://127.0.0.1:8000/submit-name", {
    method: "POST",
    headers: {
      "Content-Type": "application/json",
    },
    body: JSON.stringify({ name }),
  });

  const data = await response.json();
  console.log("Backend response:", data);
  alert(data.message);
};
```

🧩 Line-by-Line (Easy Words)

▶ `const handleSubmit = async (e) => {`

📌 Meaning:

- `handleSubmit` → function name
- `async` → this function will **wait for backend response**
- `e` → event object (form submit event)

🧠 Same idea as:

```python
async def submit_name():
```

▶ `e.preventDefault();`

📌 Meaning:

- Stops browser **page refresh**
- Keeps SPA behavior (React style)

🧠 Without this:
❌ Page reloads
❌ State lost

▶ `fetch("http://127.0.0.1:8000/submit-name", { ... })`

📌 Meaning:

- Sends request to backend
- URL → FastAPI POST API

🧠 Think of it as:

> "Call FastAPI function `submit_name`"

▶ `method: "POST"`

📌 Meaning:

- You are **sending data**
- Matches:

```python
@app.post("/submit-name")
```

▶ `headers: { "Content-Type": "application/json" }`

📌 Meaning:

- Tells backend:

> "I am sending JSON data"

🧠 Without this:
❌ Backend may not understand body

▶ `body: JSON.stringify({ name })`

📌 MOST IMPORTANT LINE 🔥

Let's break it 👇

1️⃣ `{ name }`

This is a **JavaScript object**:

```js
{ name: "Siddu" }
```

2️⃣ `JSON.stringify(...)`

Converts JS object → JSON string:

```json
{
  "name": "Siddu"
}
```

🧠 SAME as:

```python
json.dumps(...)
```

🔄 NOW CONNECT FRONTEND → BACKEND

🔹 JSON sent from React

```json
{
  "name": "Siddu"
}
```

🔹 FastAPI receives it

```python
async def submit_name(user: UserName):
```

FastAPI does:

```pgsql
JSON
 ↓
UserName schema
```

```
 ↓
Python object
```

Becomes:

```python
user = UserName(name="Siddu")
```

🔹 Access inside backend

```python
user.name   # "Siddu"
```

🔹 Stored in MongoDB

```python
collection.insert_one({"name": user.name})
```

🧠 Full Graphical Flow (Frontend → Backend → DB)

```pgsql
React Input Field
      |
      | name = "Siddu"
      ↓
JS Object
{ name: "Siddu" }
      |
      | JSON.stringify()
      ↓
JSON
{ "name": "Siddu" }
      |
      | POST /submit-name
      ↓
FastAPI
      |
      | UserName schema
      ↓
Python object
```

```
user.name = "Siddu"
     |
     ↓
MongoDB
```

🔹 Back to React (Response Handling)

▶ `const response = await fetch(...)`

📌 Meaning:

- Wait for backend to reply
- Similar to:

```python
result = await insert_one()
```

▶ `const data = await response.json();`

📌 Meaning:

- Convert backend JSON → JS object

Backend sends:

```json
{
  "message": "Data saved successfully",
  "id": "65abc..."
}
```

React gets:

```js
data.message
data.id
```

▶ `console.log("Backend response:", data);`

📌 Meaning:

- Print response in browser console

▶ `alert(data.message);`

📌 Meaning:

- Show backend message to user

🔑 One-Line Mapping (Very Important)

React (JS)	FastAPI (Python)
`{ name }`	`UserName` schema
JS object	Pydantic model
`JSON.stringify()`	JSON parsing
`await fetch()`	`await insert_one()`
`data.message`	`return { "message": ... }`

🧠 Final Mental Model

> **Frontend creates data**
> **→ Converts to JSON**
> **→ Backend converts JSON to object**
> **→ Object stored in DB**
> **→ Backend returns JSON**
> **→ Frontend converts JSON to object**

You're now thinking like a **full-stack developer** 👏

If you want next:

- Network tab debugging
- Error handling (try/catch)
- Why 422 error comes
- Why await is needed in React too

Just tell me 👍