USING FETCH API TO CREATE A REST CRUD OPERATION

1. First look at the structure of the HTML page, then make variable declarations:

- Get the HTML elements using `document.getElementById()`

Hint:

```
const form = document.getElementById('todoForm');
const todoText = document.getElementById('todoText');
const todoList = document.getElementById('todoList');
```

2. Allow for adding a todo:

- Add an event listener to the `todoForm`. When the form is submitted (button clicked), the provided asynchronous function (`async (e) => {...}`) should execute.
    - `e.preventDefault()` prevents the default form submission behavior (reloading the page).
    - `const text = todoText.value;` gets the entered text from the `todoText` input field.

Hint:

```
form.addEventListener('submit', async (e) => {
  e.preventDefault();
  const text = todoText.value;
  //the code in the next part goes here
});
```

    ◦

3. Sending a POST request (Fetch APi):

- This block should go inside the submit event above. It performs an HTTP POST request (typically to a backend server) at the `/todos` endpoint.
    - `fetch('/todos', { ... })` initiates the asynchronous request.
    - `method: 'POST'` specifies the HTTP method as POST, suitable for creating new data.
    - `headers: { ... }` sets the headers with `Content-Type: 'application/json'`, indicating JSON data in the request body.
    - `body: JSON.stringify({ text })` prepares the request body by converting the `text` variable (entered todo) to a JSON object.
    - `const response = await fetch(...)` waits for the response from the server.

- `const data = await response.json()` parses the JSON response (presumably containing the newly created todo data, including the ID).

Hint:

```
const response = await fetch('/todos', {
  method: 'POST',
  headers: {
    'Content-Type': 'application/json'
  },
  body: JSON.stringify({ text })
});
const data = await response.json();
```

4. Add a todo to the list:

- Update the `todoList` with the new todo:
  - `todoList.innerHTML += ...` appends HTML content to the existing list.
  - The innerHTML string dynamically creates an `li` element with the todo text (`data.text`), a "Delete" button calling `deleteTodo(data.id)`, and an "Edit" button calling `editTodo(data.id)`. The `data.id` from the server response is used for identification.
  - `todoText.value = '';` clears the input field for the next todo.

Hint:

```
todoList.innerHTML += `<li>${data.text} <button
onclick="deleteTodo(${data.id})">Delete</button> <button
onclick="editTodo(${data.id})">Edit</button></li>`;
todoText.value = '';
```

5. Fetch todos on load:

- This `fetchTodo` function fetches existing todos from the server. It's called initially to populate the list when the page loads.
  - `async function fetchTodos() { ... }` defines an asynchronous function.
  - Similar to the previous `fetch`, it retrieves data from the `/todos` endpoint using a GET request (presumably to retrieve all existing todos).
  - `const todos = await response.json()` parses the JSON response containing the array of todos.

Hint:

```
async function fetchTodos() {
  const response = await fetch('/todos');
  const todos = await response.json();
  //code in the part below goes here
}

fetchTodos();
```

6. Display fetched todos (inside function above):

Hint:

```
todoList.innerHTML = '';
      todos.forEach(todo => {
        const li = document.createElement('li');
        li.textContent = todo.text;
        const deleteButton =
document.createElement('button');
        deleteButton.textContent = 'Delete';
        deleteButton.addEventListener('click', () =>
deleteTodo(todo.id));
        const editButton =
document.createElement('button');
        editButton.textContent = 'Edit';
        editButton.addEventListener('click', () =>
editTodo(todo.id));
        li.appendChild(deleteButton);
        li.appendChild(editButton);
        todoList.appendChild(li);
      });
```

7. Deleting a todo:

- The `deleteTodo` function will be triggered when the "Delete" button is clicked.
- It will send an HTTP DELETE request to the server with the specified `id` to remove the todo.
- After the deletion, it will call `fetchTodos()` to update the displayed list.

Hint:

```
async function deleteTodo(id) {
  await fetch(`/todos/${id}`, { method: 'DELETE' });
  fetchTodos();
```

```
}
```

8. Editing a todo:

- The `editTodo` function will be triggered when the "Edit" button is clicked.
- It will prompt the user for a new text value using `prompt()`.
- It will send an HTTP PUT request to the server with the specified `id` and the new `text` in the request body.
- After the update, it will call `fetchTodos()` to refresh the displayed list.

Hint:

```
async function editTodo(id) {
  const newText = prompt('Enter new text for the todo');
  await fetch(`/todos/${id}`, {
    method: 'PUT',
    headers: {
      'Content-Type': 'application/json'
    },
    body: JSON.stringify({ text: newText })
  });
  fetchTodos();
}
```