

## Deliverable 2 Postman Results

# Calls

## Calls Table Starting Out:

The screenshot shows a web application with a table of emergency calls. The table has the following columns: **id** (INTEGER PRIMARY KEY AUTOINCREMENT), **caller\_name** (TEXT NOT NULL), **caller\_address** (TEXT NOT NULL), **call\_type** (TEXT NOT NULL), **crew\_assigned** (TEXT), **time\_called** (DATETIME DEFAULT), and **time\_dispatched** (DATETIME). The data rows are:

	id	caller_name	caller_address	call_type	crew_assigned	time_called	time_dispatched
1	1	John Doe	123 Main St	Power Outage	Crew Alpha	2023-10-25 22:00	2023-10-25 22:15
2	2	Jane Smith	456 Oak Ave	Water Leak	Crew Beta	2023-10-26 08:30	2023-10-26 09:00
3	3	Tom Lee	789 Pine Rd	Gas Leak	Crew Gamma	2023-10-27 12:45	2023-10-27 13:00

# Getting All Calls:

GET  Send

Tests Code Cookies

```
1 pm.test("Status code is 200", function () {
2   ... pm.response.to.have.status(200);
3 });
4
5 pm.test("Response contains data array", function () {
6   ... const jsonData = pm.response.json();
7   ... pm.expect(jsonData).to.have.property("data");
8   ... pm.expect(jsonData.data).to.be.an("array");
9 });
10
```

Test scripts are written in JavaScript, and are run after the response is received. Learn more about [tests scripts](#)

Snippets

- Get an environment variable
- Get a global variable
- Get a variable
- Get a collection variable
- Set an environment variable
- Set a global variable
- Set a collection variable
- Clear an environment variable
- Clear a global variable
- Clear a collection variable
- Send a request

Body Pretty Raw Preview JSON

```
1 {
2   "data": [
3     {
4       "id": 1,
5       "caller_name": "Jason Doe",
6       "caller_address": "246 Baker Ave",
7       "call_type": "Power Outage",
8       "crew_assigned": "Crew Alpha",
9       "time_called": "2024-10-28 12:00:00",
10      "time_dispatched": "2024-10-28 12:15:00",
11      "time_completed": "2024-10-28 13:00:00",
12      "issue_reported": "Power outage",
13      "issue_found": "Tripped breaker",
14      "dispatcher_id": 2
15    },
16    {
17      "id": 2,
18      "caller_name": "Jane Smith",
19      "caller_address": "456 Oak Ave",
20      "call_type": "Water Leak",

```

Status: 200 OK Time: 13 ms Size: 1.2 KB

GET  Send

Tests Code Cookies Test Results (2/2)

```
1 pm.test("Status code is 200", function () {
2   ... pm.response.to.have.status(200);
3 });
4
5 pm.test("Response contains data array", function () {
6   ... const jsonData = pm.response.json();
7   ... pm.expect(jsonData).to.have.property("data");
8   ... pm.expect(jsonData.data).to.be.an("array");
9 });
10
```

Test scripts are written in JavaScript, and are run after the response is received. Learn more about [tests scripts](#)

Snippets

- Get an environment variable
- Get a global variable
- Get a variable
- Get a collection variable
- Set an environment variable
- Set a global variable
- Set a collection variable
- Clear an environment variable
- Clear a global variable
- Clear a collection variable
- Send a request

Test Results (2/2) All Passed Skipped Failed

- PASS Status code is 200
- PASS Response contains data array

Status: 200 OK Time: 13 ms Size: 1.2 KB

## Get Call by ID:

The screenshot shows the REST Client interface with a GET request to `http://localhost:5000/api/calls/3`. The response is a JSON object with the following structure:

```
{
  "data": {
    "id": 3,
    "caller_name": "Tom Lee",
    "caller_address": "789 Pine Rd",
    "call_type": "Gas Leak",
    "crew_assigned": "Crew Gamma",
    "time_called": "2023-10-27 12:45",
    "time_dispatched": "2023-10-27 13:00",
    "time_completed": "2023-10-27 14:00",
    "issue_reported": "Smell of gas near stove",
    "issue_found": "Loose valve",
    "dispatcher_id": 1
  }
}
```

The screenshot shows the REST Client interface with the same GET request. The Test Results panel is expanded, showing two passed tests:

- PASS Status code is 200
- PASS Response contains call data

## Get Call by Time and Type:

GET

http://localhost:5000/api/calls-by-type-and-time?callType=Power Outage&time\_dispatched=2023-10-25 22:00&time\_completed=2023-10-26 00:00

Params

Authorization

Headers (6)

Body

Scripts

Settings

Query Params



<input checked="" type="checkbox"/>	Key	Value	Description
<input checked="" type="checkbox"/>	callType	Power Outage	
<input checked="" type="checkbox"/>	time_dispatched	2023-10-25 22:00	
<input checked="" type="checkbox"/>	time_completed	2023-10-26 00:00	
<input type="checkbox"/>	Key	Value	Description

Body

Cookies

Headers (8)

Test Results

200 OK • 42 ms • 596 B •  

Pretty

Raw

Preview

Visualize

```
{
  "data": [
    {
      "id": 1,
      "caller_name": "John Doe",
      "caller_address": "123 Main St",
      "call_type": "Power Outage",
      "crew_assigned": "Crew Alpha",
      "time_called": "2023-10-25 22:00",
      "time_dispatched": "2023-10-25 22:15",
      "time_completed": "2023-10-25 23:30",
      "issue_reported": "No power since 10 PM",
      "issue_found": "Tripped breaker",
      "dispatcher_id": 1
    }
  ]
}
```

# Get Call by Time:

GET

http://localhost:5000/api/calls-by-date?time\_dispatched=2023-10-25 22:00&time\_completed=2023-10-26 00:00

Params

Authorization

Headers (6)

Body

Scripts

Settings

Query Params


<input type="checkbox"/>	Key	Value	Description
<input type="checkbox"/>	callType	Power Outage	
<input checked="" type="checkbox"/>	time_dispatched	2023-10-25 22:00	
<input checked="" type="checkbox"/>	time_completed	2023-10-26 00:00	
<input type="checkbox"/>	Key	Value	Description

Body

Cookies

Headers (8)

Test Results

200 OK • 


Pretty

Raw

Preview

Visualize

JSON



```
1  {
2    "data": [
3      {
4        "id": 1,
5        "caller_name": "John Doe",
6        "caller_address": "123 Main St",
7        "call_type": "Power Outage",
8        "crew_assigned": "Crew Alpha",
9        "time_called": "2023-10-25 22:00",
10       "time_dispatched": "2023-10-25 22:15",
11       "time_completed": "2023-10-25 23:30",
12       "issue_reported": "No power since 10 PM",
13       "issue_found": "Tripped breaker",
14       "dispatcher_id": 1
15     }
16   ]
17 }
```

# Get Call by Type:

http://localhost:5000/api/calls-by-type?callType=Power Outage

GET

http://localhost:5000/api/calls-by-type?callType=Power Outage

Params Authorization Headers (6) Body Scripts Settings

Query Params		
<input type="checkbox"/> Key	Value	Description
<input checked="" type="checkbox"/> callType	Power Outage	
<input type="checkbox"/> time_dispatched	2023-10-25 22:00	
<input type="checkbox"/> time_completed	2023-10-26 00:00	
<input type="checkbox"/> Key	Value	Description

Body Cookies Headers (8) Test Results 200 OK

PrettyRawPreviewVisualizeJSON

```
1 {
2   "data": [
3     {
4       "id": 1,
5       "caller_name": "John Doe",
6       "caller_address": "123 Main St",
7       "call_type": "Power Outage",
8       "crew_assigned": "Crew Alpha",
9       "time_called": "2023-10-25 22:00",
10      "time_dispatched": "2023-10-25 22:15",
11      "time_completed": "2023-10-25 23:30",
12      "issue_reported": "No power since 10 PM",
13      "issue_found": "Tripped breaker",
14      "dispatcher_id": 1
15    }
16  ]
17 }
```

## Add Call:

API Test Collection / Calls / Add Calls

POST http://localhost:5000/api/calls Send

Body Code Cookies

☐ none ☐ form-data ☐ x-www-form-urlencoded ☒ raw ☐ binary ☐ GraphQL

```
1 {
2   "caller_name": "Max Tucker",
3   "caller_address": "2359 Woodville Rd",
4   "call_type": "Power Outage",
5   "crew_assigned": "Crew Delta",
6   "issue_reported": "Power Fluxuation"
7 }
8
```

Body Status: 201 Created Time: 23 ms Size: 278 B ... +

Pretty Raw Preview JSON ...

```
1 {
2   "id": 5
3 }
```

API Test Collection / Calls / Add Calls

POST http://localhost:5000/api/calls Send

Tests Code Cookies

```
1
2 pm.test("Status code is 201", function() {
3   pm.response.to.have.status(201);
4 });
5
6 pm.test("Response contains call ID", function() {
7   const jsonData = pm.response.json();
8   pm.expect(jsonData).to.have.property("id");
9 });
10
```

Test scripts are written in JavaScript, and are run after the response is received. Learn more about [tests scripts](#)

Snippets

- Get an environment variable
- Get a global variable
- Get a variable
- Get a collection variable
- Set an environment variable
- Set a global variable
- Set a collection variable
- Clear an environment variable
- Clear a global variable
- Clear a collection variable
- Send a request

Test Results (2/2) Status: 201 Created Time: 23 ms Size: 278 B ... +

All Passed Skipped Failed

- PASS Status code is 201
- PASS Response contains call ID



# Updating Call by ID:

pec.sqlite M Get All Calls Add Calls Get Call by ID Delete Calls by ID Update Calls by ID X JS createDatabase.js

APITestCollection / Calls / Update Calls by ID Save View Documentation No Environment

PUT http://localhost:5000/api/calls/1 Send

Body Code Cookies

none form-data x-www-form-urlencoded raw binary GraphQL

```
1 {
2   "caller_name": "John Doe",
3   "caller_address": "123 Main St",
4   "call_type": "Power Outage",
5   "crew_assigned": "Crew Alpha",
6   "time_called": "2023-10-25 22:00",
7   "time_dispatched": "2023-10-25 22:15",
8   "time_completed": "2023-10-25 23:30",
9   "issue_reported": "No power since 10 PM",
10  "dispatcher_id": "1"
11 }
12
13
```

Body Status: 200 OK Time: 12 ms Size: 324 B

Pretty Raw Preview JSON

```
1 {
2   "message": "Call entry updated successfully",
3   "changes": 1
4 }
```

pec.sqlite M Get All Calls Add Calls Get Call by ID Delete Calls by ID Update Calls by ID X JS createDatabase.js

APITestCollection / Calls / Update Calls by ID Save View Documentation No Environment

PUT http://localhost:5000/api/calls/1 Send

Tests Code Cookies

```
1 pm.test("Status code is 200", function () {
2   pm.response.to.have.status(200);
3 });
4
5 pm.test("Response confirms update", function () {
6   const jsonData = pm.response.json();
7   pm.expect(jsonData).to.have.property("message", "Call entry updated successfully");
8   pm.expect
```

Test scripts are written in JavaScript, and are run after the response is received. Learn more about [tests scripts](#)

Snippets

- Get an environment variable
- Get a global variable
- Get a variable
- Get a collection variable
- Set an environment variable
- Set a global variable
- Set a collection variable
- Clear an environment variable
- Clear a global variable
- Clear a collection variable
- Send a request

Test Results (2/2)

All Passed Skipped Failed

- PASS Status code is 200
- PASS Response confirms update



The screenshot shows the SQLite 3.46.1 interface. At the top, there are tabs for 'jpec.sqlite M', 'Get All Calls', 'Add Calls', 'Get Call by ID', 'Delete Calls by ID', 'Update Calls by ID', and 'createData'. Below the tabs, the breadcrumb 'call-tracker-app > server > jpec.sqlite' is visible. The main area shows a table named 'calls' with the following columns: 'id' (INTEGER PRIMARY KEY AUTOINCREMENT), 'caller\_name' (TEXT NOT NULL), 'caller\_address' (TEXT NOT NULL), 'call\_type' (TEXT NOT NULL), 'crew\_assigned' (TEXT), 'time\_called' (DATETIME DEFAULT), and 'time\_dispatched' (DATETIME). The table contains four rows of data:

	id	caller_name	caller_address	call_type	crew_assigned	time_called	time_dispatched
1	1	John Doe	123 Main St	Power Outage	Crew Alpha	2023-10-25 22:00	2023-10-25 22:15
2	2	Jane Smith	456 Oak Ave	Water Leak	Crew Beta	2023-10-26 08:30	2023-10-26 09:00
3	3	Tom Lee	789 Pine Rd	Gas Leak	Crew Gamma	2023-10-27 12:45	2023-10-27 13:00
4	5	Max Tucker	2359 Woodville Rd	Power Outage	Crew Delta	2024-10-29 00:15:07	NULL

Below the table, there is a '+' sign and a row of empty cells for adding new data.

# Delete Call by ID:

pec.sqlite M [Get All Calls](#) [Add Calls](#) [Get Call by ID](#) [Delete Calls by ID](#) [Update Calls by ID](#) JS createDatabase.js

APITestCollection / Calls / Update Calls by ID Save View Documentation No Environment

PUT http://localhost:5000/api/calls/1 Send

Body Code Cookies

none form-data x-www-form-urlencoded raw binary GraphQL

```
1 {
2   "caller_name": "John Doe",
3   "caller_address": "123 Main St",
4   "call_type": "Power Outage",
5   "crew_assigned": "Crew Alpha",
6   "time_called": "2023-10-25 22:00",
7   "time_dispatched": "2023-10-25 22:15",
8   "time_completed": "2023-10-25 23:30",
9   "issue_reported": "No power since 10 PM",
10  "dispatcher_id": "1"
11 }
12
13
```

Body Status: 200 OK Time: 12 ms Size: 324 B

Pretty Raw Preview JSON

```
1 {
2   "message": "Call entry updated successfully",
3   "changes": 1
4 }
```

pec.sqlite M [Get All Calls](#) [Add Calls](#) [Get Call by ID](#) [Delete Calls by ID](#) [Update Calls by ID](#) JS createDatabase.js

APITestCollection / Calls / Update Calls by ID Save View Documentation No Environment

PUT http://localhost:5000/api/calls/1 Send

Tests Code Cookies

```
1 pm.test("Status code is 200", function () {
2   pm.response.to.have.status(200);
3 });
4
5 pm.test("Response confirms update", function () {
6   const jsonData = pm.response.json();
7   pm.expect(jsonData).to.have.property("message", "Call entry updated successfully");
8   pm.expect
```

Test scripts are written in JavaScript, and are run after the response is received. Learn more about [tests scripts](#)

Snippets

- Get an environment variable
- Get a global variable
- Get a variable
- Get a collection variable
- Set an environment variable
- Set a global variable
- Set a collection variable
- Clear an environment variable
- Clear a global variable
- Clear a collection variable
- Send a request

Test Results (2/2)

All Passed Skipped Failed

- PASS Status code is 200
- PASS Response confirms update



# Getting All Crews:

The screenshot shows the JPEC SQLite M application interface. The top bar includes tabs for 'jpec.sqlite M', 'http://localhost:5000', 'JS createDatabase.js', and 'JS index.js'. The main area displays a GET request to 'http://localhost:5000/api/crews'. The 'Tests' panel on the left contains a script with two tests: one for status code 200 and another for the response containing a data array. The 'Body' panel on the right shows the JSON response, which is an array of three crew objects. The status bar at the top right indicates 'Status: 200 OK Time: 6 ms Size: 457 B'.

```
1 pm.test("Status code is 200", function() {
2   pm.response.to.have.status(200);
3 });
4 pm.test("Response contains data array", function() {
5   const jsonData = pm.response.json();
6   pm.expect(jsonData).to.have.property("data").that.is.an("array");
7 });
```

```
1 {
2   "data": [
3     {
4       "id": 1,
5       "crew_name": "Crew Alpha",
6       "crew_contact": "555-1234"
7     },
8     {
9       "id": 2,
10      "crew_name": "Crew Beta",
11      "crew_contact": "555-5678"
12     },
13     {
14       "id": 3,
15       "crew_name": "Crew Gamma",
16       "crew_contact": "555-9101"
17     }
18   ]
19 }
```

This screenshot shows the same JPEC SQLite M application interface, but with the 'Test Results' panel on the right. The 'Tests' panel on the left remains the same. The 'Test Results' panel shows two tests that have passed: 'Status code is 200' and 'Response contains data array'. The status bar at the top right indicates 'Status: 200 OK Time: 6 ms Size: 457 B'.

```
1 pm.test("Status code is 200", function() {
2   pm.response.to.have.status(200);
3 });
4 pm.test("Response contains data array", function() {
5   const jsonData = pm.response.json();
6   pm.expect(jsonData).to.have.property("data").that.is.an("array");
7 });
```

```
Test Results (2/2)
All Passed Skipped Failed
PASS Status code is 200
PASS Response contains data array
```

## Get Crew by ID:

GET  Send

Tests Code Cookies

```
1 pm.test("Status code is 200", function() {
2   pm.response.to.have.status(200);
3 });
4 pm.test("Response contains crew data", function() {
5   const jsonData = pm.response.json();
6   pm.expect(jsonData).to.have.property("data").that.is.an("object");
7   pm.expect(jsonData.data).to.have
```

Body Status: 200 OK Time: 6 ms Size: 335 B

Pretty Raw Preview JSON

```
1 {
2   "data": {
3     "id": 3,
4     "crew_name": "Crew Gamma",
5     "crew_contact": "555-9101"
6   }
7 }
```

GET  Send

Tests Code Cookies

```
1 pm.test("Status code is 200", function() {
2   pm.response.to.have.status(200);
3 });
4 pm.test("Response contains crew data", function() {
5   const jsonData = pm.response.json();
6   pm.expect(jsonData).to.have.property("data").that.is.an("object");
7   pm.expect(jsonData.data).to.have
```

Test Results (2/2) Status: 200 OK Time: 6 ms Size: 335 B

All Passed Skipped Failed

- PASS Status code is 200
- PASS Response contains crew data

## Add Crews:

The screenshot shows the REST Client interface with a POST request to `http://localhost:5000/api/crews`. The request body is a JSON object: `{ "crew_name": "Delta Crew", "crew_contact": "683-2464" }`. The response status is `201 Created` with a time of `12 ms` and a size of `278 B`. The response body is `{ "id": 4 }`.

API Test Collection / Crews / Add Crew

POST `http://localhost:5000/api/crews` Send

Body Code Cookies

☐ none ☐ form-data ☐ x-www-form-urlencoded ☒ raw ☐ binary ☐ GraphQL

```
1 {
2   "crew_name": "Delta Crew",
3   "crew_contact": "683-2464"
4 }
5
```

Body Status: 201 Created Time: 12 ms Size: 278 B

Pretty Raw Preview JSON

```
1 {
2   "id": 4
3 }
```

The screenshot shows the REST Client interface with the same POST request. The left pane displays test scripts in JavaScript. The right pane shows the test results, which are both `PASS`.

API Test Collection / Crews / Add Crew

POST `http://localhost:5000/api/crews` Send

Tests Code Cookies

```
1
2 pm.test("Status code
   is 201", function () {
3   pm.response.to.
     have.status
       (201);
4 });
5
6 pm.test("Response
   contains crew
     ID", function () {
7   const jsonData =
     pm.response.
       json();
8   pm.expect
     (jsonData).to.
       have.property
         ("id");
9 });
10
```

Test scripts are written in JavaScript, and are run after the response is received. Learn more about [tests scripts](#).

Snippets

- Get an environment variable
- Get a global variable
- Get a variable
- Get a collection variable
- Set an environment variable
- Set a global variable
- Set a collection variable
- Clear an environment variable
- Clear a global variable
- Clear a collection variable
- Send a request

Test Results (2/2) Status: 201 Created Time: 12 ms Size: 278 B

All Passed Skipped Failed

- PASS Status code is 201
- PASS Response contains crew ID

The screenshot shows the jpec.sqlite application interface. At the top, there's a navigation bar with tabs for 'jpec.sqlite M', 'Get All Crews', 'Get Crew by ID', 'Update Crew by ID', 'Delete Crew by ID', 'Add Crew', and a 'JS createData' button. Below the navigation bar, the breadcrumb path 'call-tracker-app > server > jpec.sqlite' is displayed. The main interface has a toolbar with 'SELECT', 'FROM', 'Crews' (selected), 'Schema', 'Query Editor', 'Auto Reload', 'Find', and 'Other Tools...'. The SQLite version '3.46.1' is shown in the top right corner. The central area displays a table with three columns: 'id INTEGER PRIMARY KEY AUTOINCREMENT', 'crew\_name TEXT NOT NULL', and 'crew\_contact TEXT'. The table contains three rows of data: '1 Crew Alpha 555-1234', '2 Crew Beta 555-5678', and '3 Delta Crew 683-2464'. A '+' button is visible on the right side of the table.

	id INTEGER PRIMARY KEY AUTOINCREMENT	crew_name TEXT NOT NULL	crew_contact TEXT
1	1	Crew Alpha	555-1234
2	2	Crew Beta	555-5678
3	4	Delta Crew	683-2464

## Updating Crew by ID:

The screenshot shows the Postman interface for a PUT request to `http://localhost:5000/api/crews/3`. The request body is a JSON object:

```
{  "crew_name": "Theta Crew",  "crew_contact": "654-2222"}
```

The response is a JSON object with the following structure:

```
{  "message": "Crew updated successfully",  "changes": 1}
```

The status is 200 OK, Time: 9 ms, Size: 318 B.

The screenshot shows the Postman interface with test scripts for the PUT request. The test scripts are written in JavaScript:

```
1 pm.test("Status code is 200", function() {
2   pm.response.to.have.status(200);
3 });
4 pm.test("Response confirms update", function() {
5   const jsonData = pm.response.json();
6   pm.expect(jsonData).to.have.property("message", "Crew updated successfully");
7 });
8
```

The test results show two tests passed:

- PASS Status code is 200
- PASS Response confirms update



The screenshot shows a web application interface for managing crew members. At the top, there is a navigation bar with several tabs: 'jpec.sqlite M X', 'Get All Crews', 'Get Crew by ID', 'http://localhost...', 'createDatabase.js', and 'index.js'. Below the navigation bar, there is a breadcrumb trail: 'call-tracker-app > server > jpec.sqlite'. The main content area displays a table with the following data:

	id INTEGER PRIMARY KEY AUTOINCREMENT	crew_name TEXT NOT NULL	crew_contact TEXT	
1	1	Crew Alpha	555-1234	
2	2	Crew Beta	555-5678	
3	3	Theta Crew	654-2222	
+				

Below the table, there is a 'SELECT' dropdown menu and a 'FROM' dropdown menu. The 'SELECT' dropdown is currently set to 'SELECT \* FROM Crews'. The 'FROM' dropdown is currently set to 'Crews'. There are also buttons for 'Schema', 'Query Editor', 'Auto Reload', 'Find', and 'Other Tools...'. The version 'SQLite 3.46.1' is displayed in the bottom right corner.

## Delete Crew by ID:

DELETE `http://localhost:5000/api/crews/3` Send

Body Code Cookies

☐ none ☐ form-data ☐ x-www-form-urlencoded ☒ raw ☐ binary ☐ GraphQL

```
1 {
2   ... "crew_name": "Theta Crew",
3   ... "crew_contact": "654-2222"
4 }
5
```

Body Status: 200 OK Time: 11 ms Size: 318 B

Pretty Raw Preview JSON

```
1 {
2   "message": "Crew deleted successfully",
3   "changes": 1
4 }
```

DELETE `http://localhost:5000/api/crews/3` Send

Tests Code Cookies

```
1 pm.test("Status code is 200", function () {
2   ... pm.response.to.have.status(200);
3 });
4 pm.test("Response confirms deletion", function () {
5   ... const jsonData = pm.response.json();
6   ... pm.expect(jsonData).to.have.property("message", "Crew deleted successfully");
7 });
8
```

Test scripts are written in JavaScript, and are run after the response is received. Learn more about [tests scripts](#)

Snippets

- Get an environment variable
- Get a global variable
- Get a variable
- Get a collection variable
- Set an environment variable
- Set a global variable
- Set a collection variable
- Clear an environment variable
- Clear a global variable
- Clear a collection variable
- Send a request

Test Results (2/2) Status: 200 OK Time: 11 ms Size: 318 B

All Passed Skipped Failed

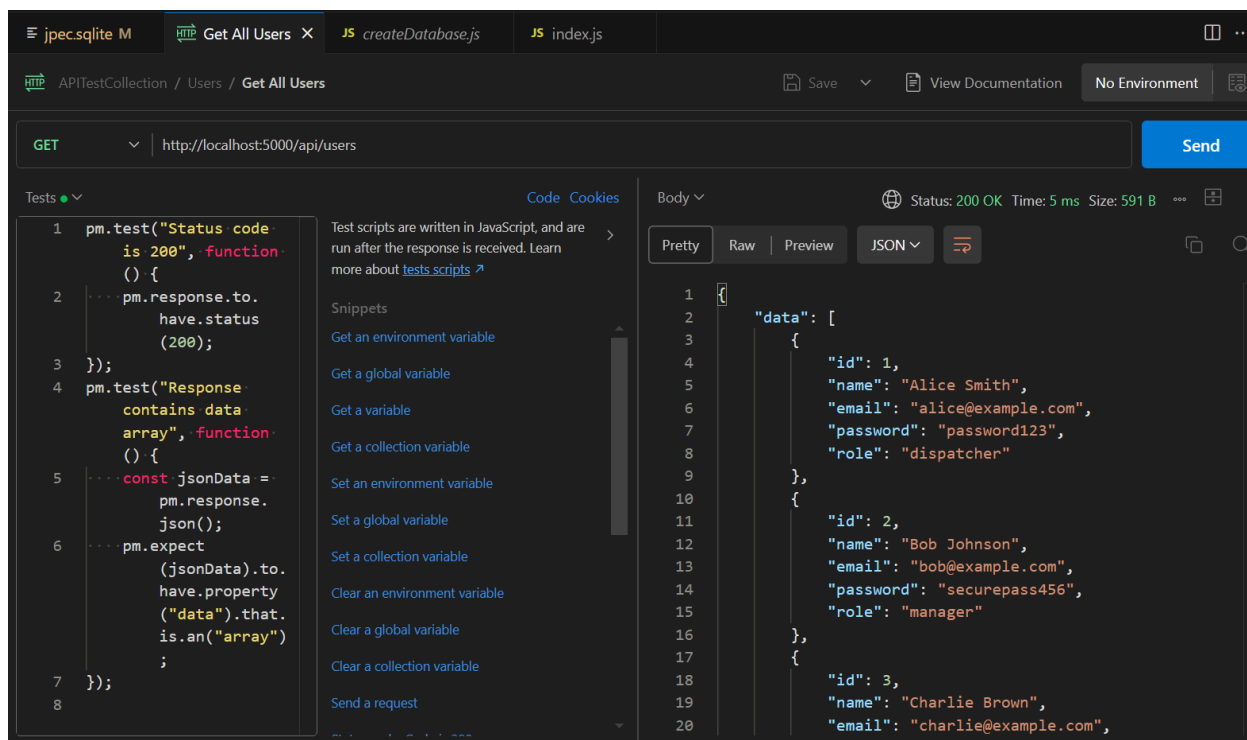
- PASS Status code is 200
- PASS Response confirms deletion

[illegible]

# Users

## User Table Starting Out:

# Getting All Users:



GET  Send

Tests Code Cookies

```
1 pm.test("Status code is 200", function() {
2   pm.response.to.have.status(200);
3 });
4 pm.test("Response contains data array", function() {
5   const jsonData = pm.response.json();
6   pm.expect(jsonData).to.have.property("data").that.is.an("array");
7 });
8
```

Test scripts are written in JavaScript, and are run after the response is received. Learn more about [tests scripts](#)

Snippets

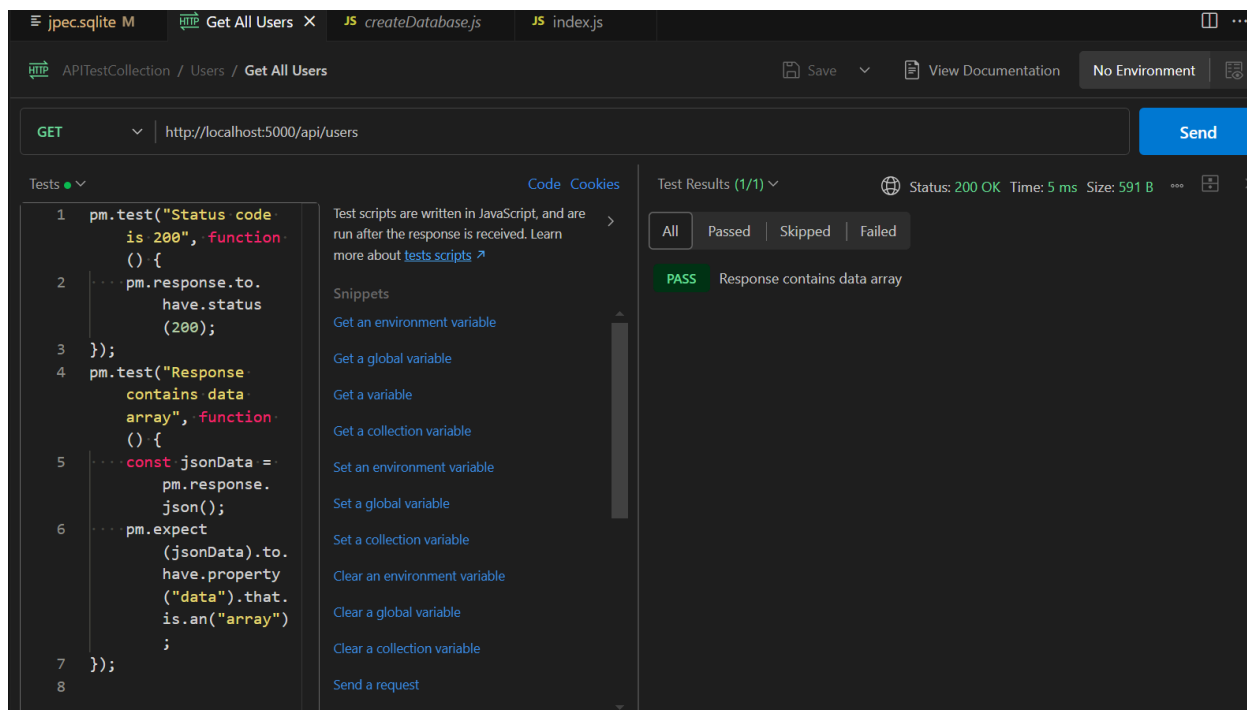
- Get an environment variable
- Get a global variable
- Get a variable
- Get a collection variable
- Set an environment variable
- Set a global variable
- Set a collection variable
- Clear an environment variable
- Clear a global variable
- Clear a collection variable
- Send a request

Body Pretty Raw Preview JSON Send

```
1 {
2   "data": [
3     {
4       "id": 1,
5       "name": "Alice Smith",
6       "email": "alice@example.com",
7       "password": "password123",
8       "role": "dispatcher"
9     },
10    {
11      "id": 2,
12      "name": "Bob Johnson",
13      "email": "bob@example.com",
14      "password": "securepass456",
15      "role": "manager"
16    },
17    {
18      "id": 3,
19      "name": "Charlie Brown",
20      "email": "charlie@example.com",

```

Status: 200 OK Time: 5 ms Size: 591 B



GET  Send

Tests Code Cookies Test Results (1/1)

```
1 pm.test("Status code is 200", function() {
2   pm.response.to.have.status(200);
3 });
4 pm.test("Response contains data array", function() {
5   const jsonData = pm.response.json();
6   pm.expect(jsonData).to.have.property("data").that.is.an("array");
7 });
8
```

Test scripts are written in JavaScript, and are run after the response is received. Learn more about [tests scripts](#)

Snippets

- Get an environment variable
- Get a global variable
- Get a variable
- Get a collection variable
- Set an environment variable
- Set a global variable
- Set a collection variable
- Clear an environment variable
- Clear a global variable
- Clear a collection variable
- Send a request

Test Results (1/1) All Passed Skipped Failed

**PASS** Response contains data array

Status: 200 OK Time: 5 ms Size: 591 B

# Getting User by ID:

The screenshot shows the Postman interface with a GET request to `http://localhost:5000/api/users/2`. The response status is `200 OK` with a time of `9 ms` and a size of `376 B`. The response body is a JSON object:

```
{  "data": {    "id": 2,    "name": "Bob Johnson",    "email": "bob@example.com",    "password": "securepass456",    "role": "manager"  }}
```

The Tests tab on the left contains the following JavaScript code:

```
1 pm.test("Status code is 200", function() {
2   pm.response.to.have.status(200);
3 });
4 pm.test("Response contains user data", function() {
5   const jsonData = pm.response.json();
6   pm.expect(jsonData).to.have.property("data").that.is.an("object");
7   pm.expect(jsonData.data).to.have.
```

The screenshot shows the same Postman interface, but now the Test Results tab is active. The test results show a `PASS` status for the test `Response contains user data`. The test results are displayed as follows:

```
Test Results (1/1)
All Passed Skipped Failed
PASS Response contains user data
```

The Tests tab on the left contains the same JavaScript code as in the previous screenshot:

```
1 pm.test("Status code is 200", function() {
2   pm.response.to.have.status(200);
3 });
4 pm.test("Response contains user data", function() {
5   const jsonData = pm.response.json();
6   pm.expect(jsonData).to.have.property("data").that.is.an("object");
7   pm.expect(jsonData.data).to.have.
```

## Add User:

The screenshot shows the Postman interface with a POST request to `http://localhost:5000/api/users`. The request body is a JSON object: `{ "name": "Dale Presto", "email": "dpresto@example.com", "password": "expresso23!", "role": "dispatcher" }`. The response status is 201 Created, with a time of 13 ms and a size of 278 B. The response body is `{ "id": 4 }`.

The screenshot shows the Postman interface with the same POST request. The left pane displays test scripts written in JavaScript. The right pane shows the test results, which are all passed.

**Test Scripts:**

```
1 pm.test("Status code is 201", function() {
2   pm.response.to.have.status(201);
3 });
4 pm.test("Response contains user ID", function() {
5   const jsonData = pm.response.json();
6   pm.expect(jsonData).to.have.property("id");
7 });
8
```

**Test Results (2/2):**

- PASS Status code is 201
- PASS Response contains user ID

The screenshot shows the SQLite Studio application. At the top, there's a toolbar with icons for file operations and a menu bar. Below the menu bar, the current project is 'call-tracker-app' and the active database is 'server'. The 'Users' table is selected in the 'FROM' dropdown. The table structure is displayed in the main area, showing columns: id (INTEGER PRIMARY KEY AUTOINCREMENT), name (TEXT NOT NULL), email (TEXT NOT NULL UNIQUE), password (TEXT NOT NULL), and role (TEXT NOT NULL). The table contains four rows of data:

	id	name	email	password	role
1	1	Alice Smith	alice@example.com	password123	dispatcher
2	2	Bob Johnson	bob@example.com	securepass456	manager
3	3	Charlie Brown	charlie@example.com	mypassword789	dispatcher
4	4	Dale Presto	dpresto@example.com	expresso23!	dispatcher



# Updating User by ID:

The screenshot shows the REST Client interface with a PUT request to `http://localhost:5000/api/users/2`. The request body is a JSON object representing a user update. The response is a JSON object indicating the update was successful.

**Request:**

```
PUT http://localhost:5000/api/users/2
```

**Body (raw):**

```
1 {
2   "name": "Yancy Becket",
3   "email": "ybecket@example.com",
4   "password": "GD@ngerMK3",
5   "role": "dispatcher"
6 }
```

**Response (JSON):**

```
1 {
2   "message": "User updated successfully",
3   "changes": 1
4 }
```

Status: 200 OK Time: 11 ms Size: 318 B

The screenshot shows the REST Client interface with test scripts for the PUT request. The test scripts are written in JavaScript and verify the response status and message. The test results show that the tests passed.

**Tests:**

```
1 pm.test("Status code is 200", function () {
2   pm.response.to.have.status(200);
3 });
4 pm.test("Response confirms update", function () {
5   const jsonData = pm.response.json();
6   pm.expect(jsonData).to.have.property("message", "User updated successfully");
7 });
8
```

**Test Results (1/1):**

- Passed: Response confirms update

Status: 200 OK Time: 11 ms Size: 318 B

The screenshot shows the SQLite CLI interface with the 'Users' table selected. The table structure is as follows:

id	name	email	password	role
1	Alice Smith	alice@example.com	password123	dispatcher
2	Yancy Becket	ybecket@example.com	G0@ngernK3	dispatcher
3	Charlie Brown	charlie@example.com	mypassword789	dispatcher
4	Dale Presto	dpresto@example.com	expresso23!	dispatcher

## Deleting User by ID:

DELETE `http://localhost:5000/api/users/4` Send

Body Code Cookies

☐ none ☐ form-data ☐ x-www-form-urlencoded ☒ raw ☐ binary ☐ GraphQL

```
1 {
2   "name": "Dale Presto",
3   "email": "dpresto@example.com",
4   "password": "expresso23!",
5   "role": "dispatcher"
6 }
```

Body Status: 200 OK Time: 12 ms Size: 318 B

Pretty Raw Preview JSON

```
1 {
2   "message": "User deleted successfully",
3   "changes": 1
4 }
```

DELETE `http://localhost:5000/api/users/4` Send

Tests Code Cookies

```
1 pm.test("Status
   code is 200",
   function () {
2     pm.response.to.
       have.status
       (200);
3   });
4 pm.test("Response
   confirms
   deletion",
   function () {
5     const jsonData
       = pm.
         response.
         json();
6     pm.expect
       (jsonData).
       to.have.
       property
       ("message",
       "User
       deleted
       successfully");
```

Test Results (2/2) Status: 200 OK Time: 12 ms Size: 318 B

All Passed Skipped Failed

- PASS Status code is 200
- PASS Response confirms deletion

The screenshot shows the SQLite Studio interface. At the top, there's a menu bar with options like 'File', 'Edit', 'View', 'Tools', 'Window', and 'Help'. Below the menu bar is a toolbar with icons for various actions. The main window displays the 'call-tracker-app' database, and the 'Users' table is selected. The table structure is shown in the top pane, and the data rows are listed in the bottom pane. The data rows are:

id	name	email	password	role
1	Alice Smith	alice@example.com	password123	dispatcher
2	Yancy Becket	ybecket@example.com	GD@ngerMK3	dispatcher
3	Charlie Brown	charlie@example.com	mypassword789	dispatcher