

# TESTING for APIs using POSTMAN

## DATABASE: MongoDB

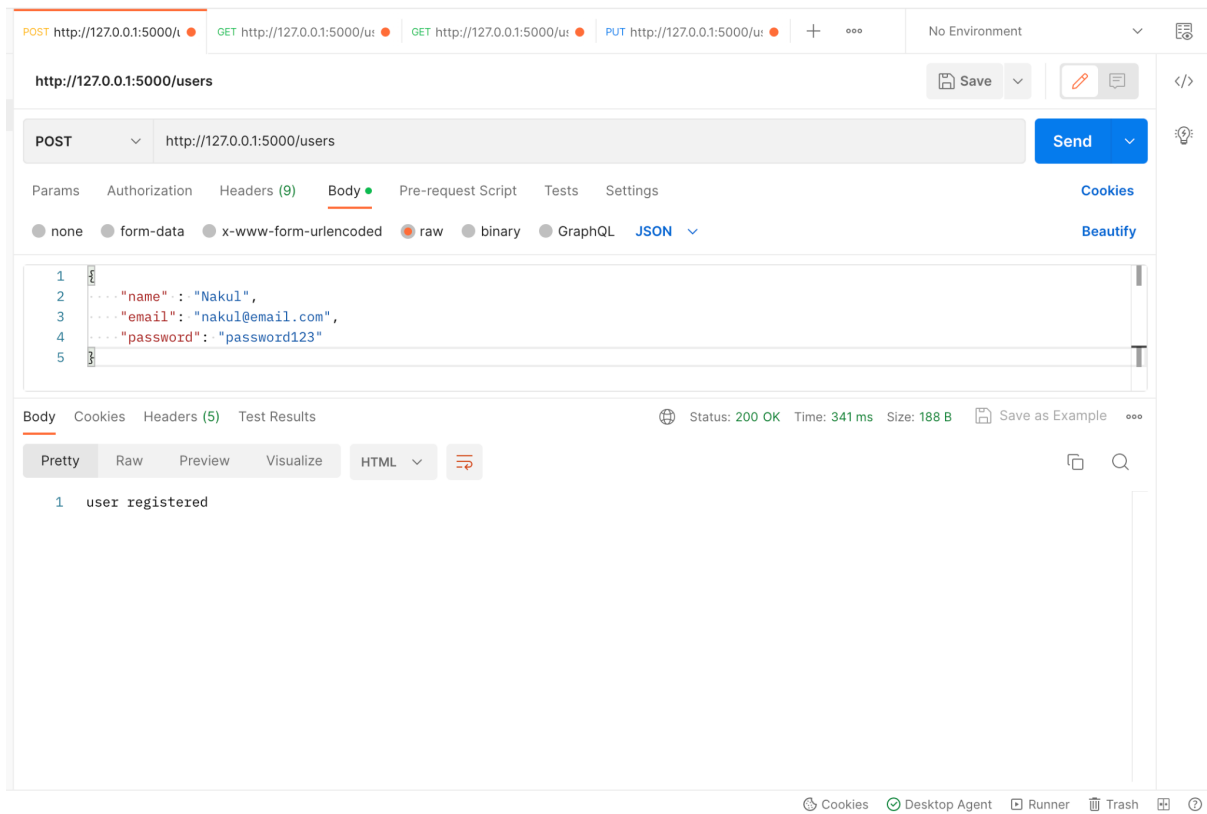
The screenshot shows the MongoDB Atlas web interface. On the left, there's a sidebar with navigation options like 'DEPLOYMENT', 'Database', 'Data Lake', 'SERVICES', 'Triggers', 'Data API', 'Data Federation', 'Search', 'SECURITY', 'Database Access', 'Network Access', 'Advanced', and 'Oto'. The main panel displays the 'db.users' collection. It shows statistics: STORAGE SIZE: 36KB, LOGICAL DATA SIZE: 852B, TOTAL DOCUMENTS: 6, INDEXES TOTAL SIZE: 36KB. Below this, there's a 'Find' tab with a filter bar showing '{ field: 'value' }'. The document list shows three users: 'Sparshh', 'Shivani', and 'Ashish', each with their respective IDs, emails, and passwords (represented as BINARY data).

### 1. GET /users - Returns a list of all users.

The screenshot shows the Postman interface. At the top, there's a toolbar with various request types (GET, POST, PUT, DELETE) and a 'Send' button. The main panel shows a GET request to 'http://127.0.0.1:5000/users'. The response is displayed in the 'Body' tab, showing a JSON array of two user objects. The first user is 'Sparshh' and the second is 'Shivani'. The response status is '200 OK', the time is '47 ms', and the size is '1.44 KB'. The bottom of the interface shows a status bar with 'Cookies', 'Desktop Agent', 'Runner', 'Trash', and a help icon.

<Returns all the records from the users collection>

## 2. POST /users - Creates a new user with the specified data



POST http://127.0.0.1:5000/users

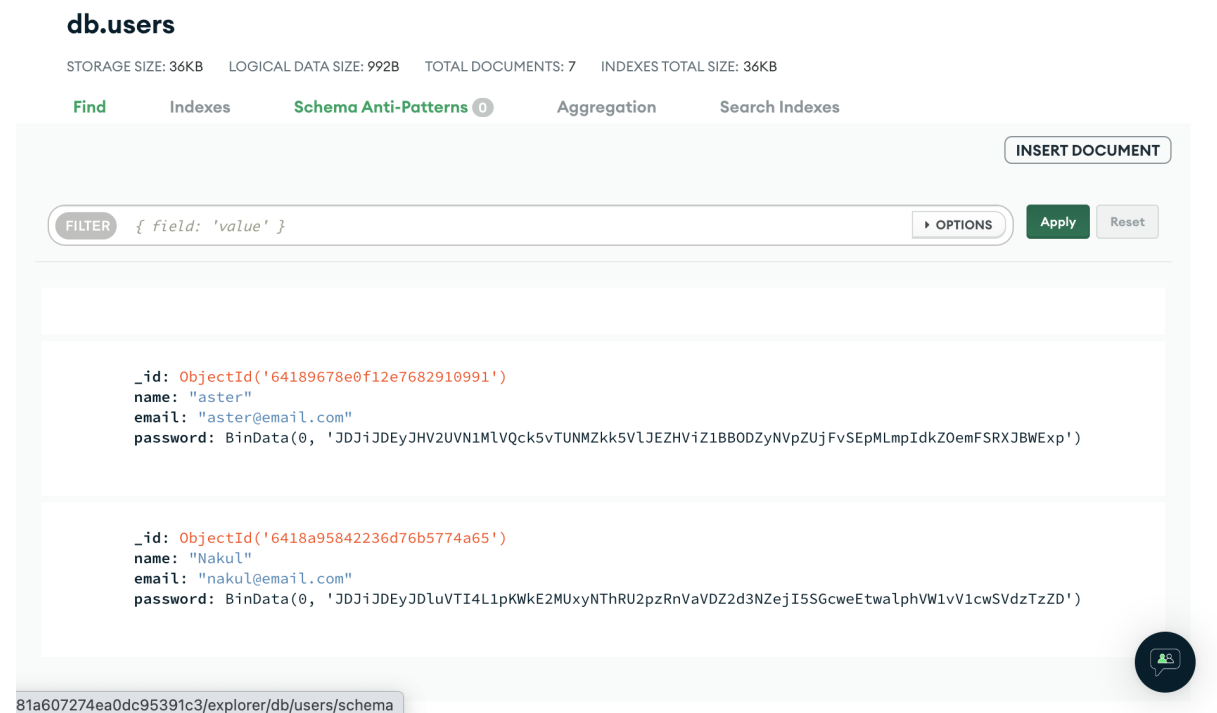
Body

```
1 {
2   "name": "Nakul",
3   "email": "nakul@email.com",
4   "password": "password123"
5 }
```

Status: 200 OK Time: 341 ms Size: 188 B

Body

```
1 user registered
```



db.users

STORAGE SIZE: 36KB LOGICAL DATA SIZE: 992B TOTAL DOCUMENTS: 7 INDEXES TOTAL SIZE: 36KB

Find Indexes Schema Anti-Patterns 0 Aggregation Search Indexes

INSERT DOCUMENT

FILTER { field: 'value' } OPTIONS Apply Reset

```
_id: ObjectId('64189678e0f12e7682910991')
name: "aster"
email: "aster@email.com"
password: BinData(0, 'JDJiJDEyJHV2UVN1MlVQck5vTUNMZkk5VlJEZHViZ1BBODZyNVpZUjFvSEpMLmpIdkZOemFSRXJBWExp')
```

```
_id: ObjectId('6418a95842236d76b5774a65')
name: "Nakul"
email: "nakul@email.com"
password: BinData(0, 'JDJiJDEyJDluVTI4L1pKWkE2MUxyNThRU2pzRnVaVDZ2d3NZejI5SGcweEtwa'lpVW1vV1cwSVdzTzZD')
```

81a607274ea0dc95391c3/explorer/db/users/schema

<Adds record to the users collection>

### 3. GET /users/<id> - Returns the user with the specified ID.

The screenshot shows a REST client interface with a GET request to `http://127.0.0.1:5000/users/64189136cde76d51c026a145`. The response status is 200 OK, with a time of 21 ms and a size of 387 B. The response body is displayed in JSON format, showing a user object with the following fields:

```
{
  "_id": {
    "$oid": "64189136cde76d51c026a145"
  },
  "email": "shivani@email.com",
  "name": "Shivani",
  "password": {
    "$binary": {
      "base64": "JDJiJDYyJHVCbGgvaUtwMm1xR0FiUjJhZS51dEFHb1p1MkpKV2N0cDhXbGdiR3Bpa3JlUx1",
      "subType": "00"
    }
  }
}
```

<Returns record on passing id in flask route through GET request>

### 4. PUT /users/<id> - Updates the user with the specified ID with the new data.

The screenshot shows a REST client interface with a PUT request to `http://127.0.0.1:5000/users/6418a95842236d76b5774a65`. The request body is in JSON format, showing a user object with the following fields:

```
{
  "name": "nakull",
  "email": "nakull@email.com",
  "password": "password321"
}
```

The response status is 200 OK, with a time of 316 ms and a size of 185 B. The response body is displayed in HTML format, showing the text "user updated".

**db.users**

STORAGE SIZE: 36KB   LOGICAL DATA SIZE: 994B   TOTAL DOCUMENTS: 7   INDEXES TOTAL SIZE: 36KB

Find   Indexes   **Schema Anti-Patterns** ①   Aggregation   Search Indexes

INSERT DOCUMENT

FILTER { field: 'value' }   ▶ OPTIONS   Apply   Reset

```

_id: ObjectId('64189678e0f12e7682910991')
name: "aster"
email: "aster@email.com"
password: BinData(0, 'JDDjiJDEyJHV2UVN1MLVQck5vTUNMZkk5VLJEZHViz1BBODZyNVpZUjFvSEpMLmpIdkZ0emFSRXJBWExp')

_id: ObjectId('6418a95842236d76b5774a65')
name: "nakull"
email: "nakull@email.com"
password: BinData(0, 'JDDjiJDEyJFp60DZVNS42TWwQUlRbVY5UGR5dGVNL3J3VvV1V2tmQlRoUnUveTFmanh2TVLRRGFVaTMy')

```

31a607274ea0dc95391c3/explorer/db/users/schema

<Using data of user we registered earlier. User record updated and changes reflected in mongoDB>

5. DELETE /users/<id> - Deletes the user with the specified ID.

http://127.0.0.1:5000/users/64188e7d8f394c497ae75ae6   Save   Send

DELETE   http://127.0.0.1:5000/users/64188e7d8f394c497ae75ae6   Send

Params   Authorization   Headers (7)   Body   Pre-request Script   Tests   Settings   Cookies

Query Params

Key	Value	Description	...	Bulk Edit
-----	-------	-------------	-----	-----------

Body   Cookies   Headers (5)   Test Results   Status: 200 OK   Time: 65 ms   Size: 179 B   Save as Example

Pretty   Raw   Preview   Visualize   HTML   1   deleted

Cookies   Desktop Agent   Runner   Trash

<Deleted record from collection by passing id in flask route through DELETE request>