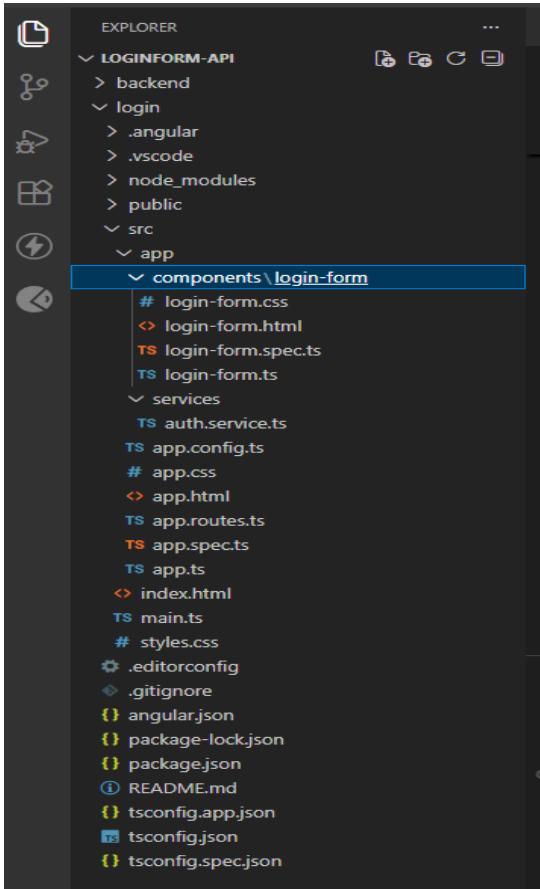


Practical No : 05

Aim : - Connecting Angular Login Form to rest API

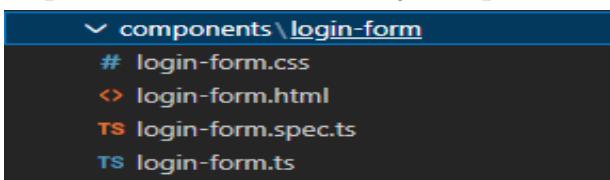
Step 1 : Creating a new file



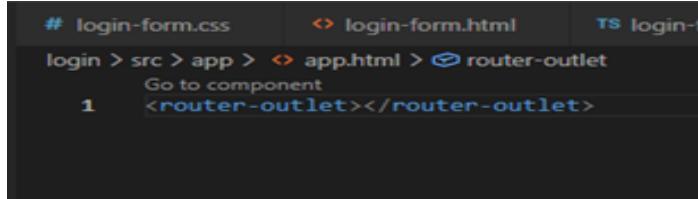
Step 2 : Open the terminal and run the required commands

```
C:\Users\Anish\Documents\MeanStack\login>ng g c components/login-form
Node.js version v25.2.1 detected.
Odd numbered Node.js versions will not enter LTS status and should not be
dejs.org/en/about/previous-releases/.
CREATE src/app/components/login-form/login-form.spec.ts (576 bytes)
CREATE src/app/components/login-form/login-form.ts (212 bytes)
CREATE src/app/components/login-form/login-form.css (0 bytes)
CREATE src/app/components/login-form/login-form.html (26 bytes)
```

Step 3 : Create the necessary components

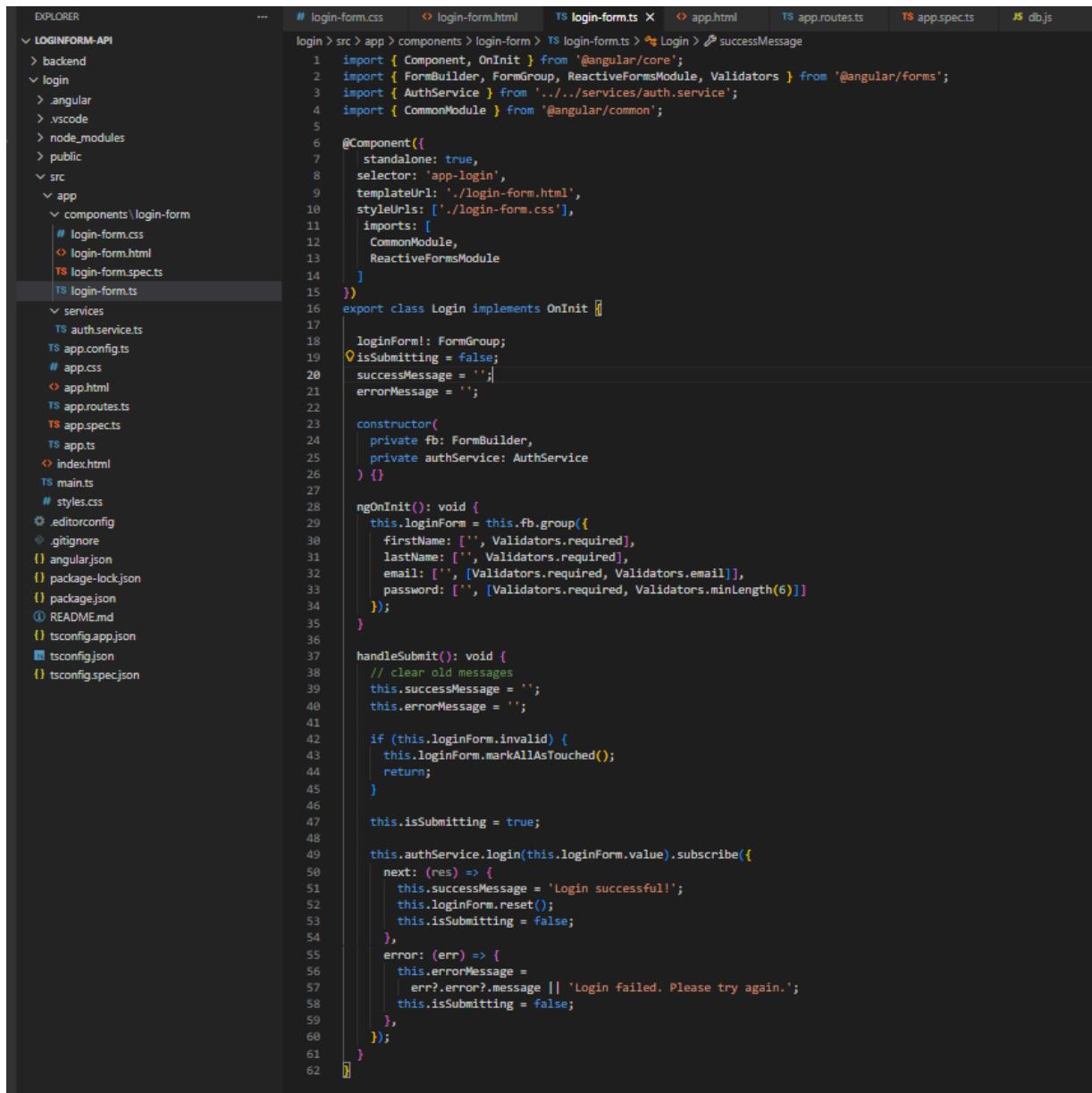


Step 4: Edit the app.html file to define the main structure



```
# login-form.css      login-form.html      TS login-f
login > src > app > app.html > router-outlet
    Go to component
1   <router-outlet></router-outlet>
```

Step 5 : Edit the login-form.ts file to write the logic



```
EXPLORER          ... # login-form.css      login-form.html      TS login-form.ts X      app.html      TS app.routes.ts      TS app.spec.ts      JS db.js
✓ LOGINFORM-API
  > backend
  ✓ login
    > .angular
    > .vscode
    > node_modules
    > public
  ✓ src
    ✓ app
      ✓ components login-form
        # login-form.css
        login-form.html
        TS login-form.spec.ts
        TS login-form.ts
      ✓ services
        TS auth.service.ts
        TS app.config.ts
      # app.css
      app.html
      TS app.routes.ts
      TS app.spec.ts
      TS app.ts
      index.html
      TS main.ts
      # styles.css
      .editorconfig
      .gitignore
      angular.json
      package-lock.json
      package.json
      README.md
      tsconfig.app.json
      tsconfig.json
      tsconfig.spec.json
```

```
login > src > app > components > login-form > TS login-form.ts > Login > successMessage
1   import { Component, OnInit } from '@angular/core';
2   import { FormBuilder, FormGroup, ReactiveFormsModule, Validators } from '@angular/forms';
3   import { AuthService } from '../../../../../services/auth.service';
4   import { CommonModule } from '@angular/common';
5
6   @Component({
7     standalone: true,
8     selector: 'app-login',
9     templateUrl: './login-form.html',
10    styleUrls: ['./login-form.css'],
11    imports: [
12      CommonModule,
13      ReactiveFormsModule
14    ]
15  })
16  export class Login implements OnInit {
17
18    loginForm!: FormGroup;
19    isSubmitting = false;
20    successMessage = '';
21    errorMessage = '';
22
23    constructor(
24      private fb: FormBuilder,
25      private authService: AuthService
26    ) {}
27
28    ngOnInit(): void {
29      this.loginForm = this.fb.group({
30        firstName: ['', Validators.required],
31        lastName: ['', Validators.required],
32        email: ['', [Validators.required, Validators.email]],
33        password: ['', [Validators.required, Validators.minLength(6)]]
34      });
35    }
36
37    handleSubmit(): void {
38      // clear old messages
39      this.successMessage = '';
40      this.errorMessage = '';
41
42      if (this.loginForm.invalid) {
43        this.loginForm.markAllAsTouched();
44        return;
45      }
46
47      this.isSubmitting = true;
48
49      this.authService.login(this.loginForm.value).subscribe({
50        next: (res) => {
51          this.successMessage = 'Login successful!';
52          this.loginForm.reset();
53          this.isSubmitting = false;
54        },
55        error: (err) => {
56          this.errorMessage =
57            err?.error?.message || 'Login failed. Please try again.';
58          this.isSubmitting = false;
59        },
60      });
61    }
62 }
```

Step 6 : Write the HTML code for the login form in login-form.html.

The screenshot shows the VS Code interface with the Explorer, Editor, and Search panes visible. The Explorer pane shows the project structure under 'LOGINFORM-API'. The Editor pane displays the content of 'login-form.html'.

```

<div class="login-form-container">
  <form [formGroup]="loginForm" (ngSubmit)="handleSubmit()">
    <h1>LOGIN FORM</h1>

    <!-- FIRST NAME -->
    <div class="form-group">
      <label>First Name</label>
      <input
        class="login-input"
        type="text"
        formControlName="firstName"
        placeholder="Enter first name"
      />

      <small class="error" *ngIf="loginForm.get('firstName')?.touched && loginForm.get('firstName')?.errors?.['required']">
        First name is required
      </small>
    </div>

    <!-- LAST NAME -->
    <div class="form-group">
      <label>Last Name</label>
      <input
        class="login-input"
        type="text"
        formControlName="lastName"
        placeholder="Enter last name"
      />

      <small class="error" *ngIf="loginForm.get('lastName')?.touched && loginForm.get('lastName')?.errors?.['required']">
        Last name is required
      </small>
    </div>

    <!-- EMAIL -->
    <div class="form-group">
      <label>Email</label>
      <input
        class="login-input"
        type="email"
        formControlName="email"
        placeholder="Enter email"
      />

      <small class="error" *ngIf="loginForm.get('email')?.touched && loginForm.get('email')?.errors?.['required']">
        Email is required
      </small>

      <small class="error" *ngIf="loginForm.get('email')?.touched && loginForm.get('email')?.errors?.['email']">
        Enter a valid email address
      </small>
    </div>

    <!-- PASSWORD -->
    <div class="form-group">
      <label>Password</label>
      <input
        class="login-input"
        type="password"
        formControlName="password"
        placeholder="Enter password"
      />

      <small class="error" *ngIf="loginForm.get('password')?.touched && loginForm.get('password')?.errors?.['required']">
        Password is required
      </small>

      <small class="error" *ngIf="loginForm.get('password')?.touched && loginForm.get('password')?.errors?.['minlength']">
        Password must be at least 6 characters
      </small>
    </div>

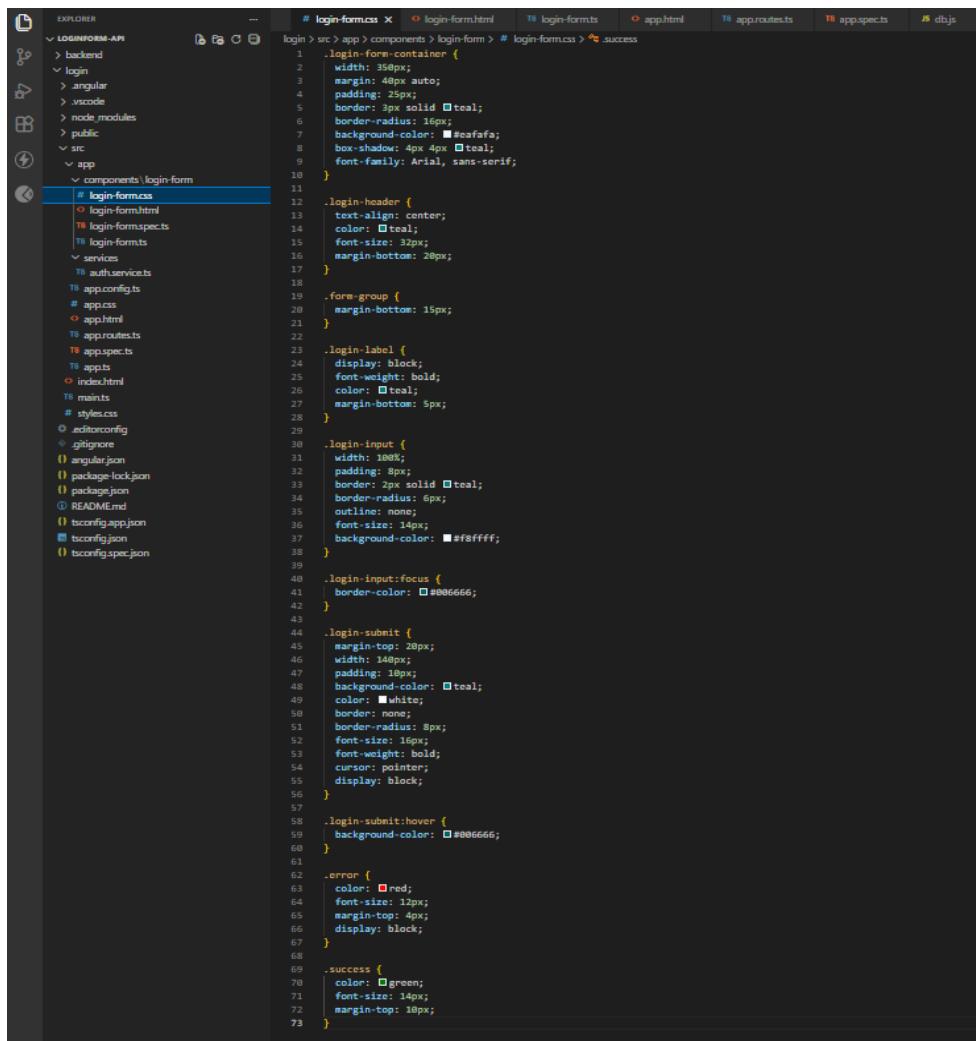
    <!-- SUBMIT BUTTON -->
    <button
      class="login-submit"
      type="submit"
      [disabled]="isSubmitting"
      {{ isSubmitting ? 'Submitting...' : 'Submit' }}
    </button>

    <!-- SUCCESS MESSAGE -->
    <p class="success" *ngIf="successMessage">
      {{ successMessage }}
    </p>

    <!-- ERROR MESSAGE -->
    <p class="error" *ngIf="errorMessage">
      {{ errorMessage }}
    </p>
  </form>
</div>

```

Step 7 : Write the CSS code for styling the login form in login-form.css.



```
# login-form.css
login > app > components > login-form > # login-form.css > success

.login-form-container {
    width: 350px;
    margin: 40px auto;
    padding: 25px;
    border: 2px solid teal;
    border-radius: 15px;
    background-color: #eafafa;
    box-shadow: 4px 4px teal;
    font-family: Arial, sans-serif;
}

.login-header {
    text-align: center;
    color: teal;
    font-size: 32px;
    margin-bottom: 20px;
}

.form-group {
    margin-bottom: 15px;
}

.login-label {
    display: block;
    font-weight: bold;
    color: teal;
    margin-bottom: 5px;
}

.login-input {
    width: 100%;
    padding: 8px;
    border: 2px solid teal;
    border-radius: 6px;
    outline: none;
    font-size: 14px;
    background-color: #f8ffff;
}

.login-input:focus {
    border-color: #006666;
}

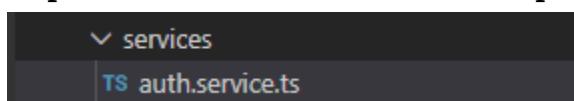
.login-submit {
    margin-top: 20px;
    width: 140px;
    padding: 10px;
    background-color: teal;
    color: white;
    border: none;
    border-radius: 8px;
    font-size: 16px;
    font-weight: bold;
    cursor: pointer;
    display: block;
}

.login-submit:hover {
    background-color: #006666;
}

.error {
    color: red;
    font-size: 12px;
    margin-top: 4px;
    display: block;
}

.success {
    color: green;
    font-size: 14px;
    margin-top: 16px;
}
```

Step 8: Create a services folder in the project



Step 9: Create and write code in auth.service.ts

```

EXPLORER
... Student.js authRoutes.js studentRoutes.js package-lock.json package.json .env
 LOGINFORM-API
 > backend
 > login
 > .angular
 > .vscode
 > node_modules
 > public
 > src
 > app
 > components\login-form
 # login-form.css
 < login-form.html
 TS login-form.spec.ts
 TS login-form.ts
 > services
 < authservice.ts

```

```

1 import { Injectable } from '@angular/core';
2 import { HttpClient } from '@angular/common/http';
3 import { Observable } from 'rxjs';
4
5 @Injectable({
6   providedIn: 'root'
7 })
8 export class AuthService {
9
10 private API_URL = 'http://localhost:3000/api/auth/signup'; // change if needed
11
12 constructor(private http: HttpClient) {}
13
14 login(data: any): Observable<any> {
15   return this.http.post(this.API_URL, data);
16 }
17

```

Step 10: Create a separate folder for the backend.

```

LOGINFOM-API
backend
 > config
 > models
 > node_modules
 > routes
 .env
 JS app.js
 {} package-lock.json
 {} package.json

```

```

1 const mongoose = require("mongoose");
2
3 const connectDB = async () => {
4   try {
5     await mongoose.connect(process.env.MONGO_URI);
6     console.log("MongoDB Connected");
7   } catch (error) {
8     console.error(error.message);
9     process.exit(1);
10 }
11
12 module.exports = connectDB;
13
14

```

```

backend
 config
 > db.js
 > models
 > node_modules
 > routes
 .env
 JS app.js
 {} package-lock.json
 {} package.json

```

```

1 const { request } = require("express");
2 const mongoose = require("mongoose");
3
4 const studentSchema = new mongoose.Schema({
5   firstName: { type: String, required: true },
6   lastName: { type: String, required: true },
7   email: { type: String, required: true },
8   password: { type: String, required: true }
9 });
10
11 module.exports = mongoose.model("Student", studentSchema);
12
13

```

```

backend
> config
> models
> node_modules
> routes
  > authRoutes.js
  > studentRoutes.js
  .env
  app.js
  package-lock.json
  package.json
> login

```

```

1 const express = require("express");
2 const Student = require("../models/student");
3 const router = express.Router();
4
5 router.post("/signup", async (req, res) => {
6   try {
7     const student = await Student.create(req.body);
8     res.status(201).json(student);
9   } catch (err) {
10     res.status(400).json({ error: err.message });
11   }
12 }
13 );
14
15 router.post("/login", async (req, res) => {
16   try {
17     const { email, password } = req.body;
18     const student = await Student.findOne({
19       email: email,
20       password: password
21     }).select({
22       password: 0
23     });
24
25     if (!student) {
26       res.status(400).json({
27         message: "invalid email or password"
28       });
29     }
30
31     res.status(201).json({
32       data: student
33     });
34   } catch (err) {
35     res.status(400).json({ error: err.message });
36   }
37 });
38
39
40
41 module.exports = router;

```

```

backend
> config
> models
> node_modules
> routes
  > authRoutes.js
  > studentRoutes.js
  .env
  app.js
  package-lock.json
  package.json
> login

```

```

1 const express = require("express");
2 const Student = require("../models/student");
3 const router = express.Router();
4
5
6 router.get("/", async (req, res) => {
7   const students = await Student.find()
8     .select({
9       password: 0
10     });
11   res.json(
12     {
13       data: students
14     }
15   );
16 }
17 );
18
19 module.exports = router;

```

Step 11: Create a .env file to store environment variables.

```

LOGINFORM-API
  backend
    > config
    > models
    > node_modules
    > routes
      > authRoutes.js
      > studentRoutes.js
    .env
    app.js
    package-lock.json
    package.json
  > login

```

```

backend > .env
1 PORT=3000
2 MONGO_URI=mongodb://127.0.0.1:27017/collegeDB
3

```

Step 12: Create an app.js file for backend server logic.

The screenshot shows the file structure of the `LoginForm-API` project. The `backend` folder contains `config`, `models`, `node_modules`, `routes`, `authRoutes.js`, `studentRoutes.js`, `.env`, and `app.js`. The `app.js` file is selected and its code is displayed:

```

1  const express = require("express");
2  const dotenv = require("dotenv");
3  const connectDB = require("./config/db");
4  const cors = require("cors");
5  dotenv.config();
6  connectDB();
7
8  const app = express();
9  app.use(express.json());
10 app.use(cors());
11
12 app.use("/api/auth", require("./routes/authRoutes"));
13 app.use("/api/student", require("./routes/studentRoutes"));
14
15 const PORT = process.env.PORT || 5000;
16 app.listen(PORT, () => console.log(`Server running on port ${PORT}`));
17

```

Step 13: Run the backend server

The terminal window shows the command `node app.js` being run and the output indicating the server is running on port 3000 and MongoDB is connected.

```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS POSTMAN CONSOLE

● PS C:\Users\Admin\Documents\SEM-1V\MeanStack\Module 1\LoginForm-API> cd backend
○ PS C:\Users\Admin\Documents\SEM-1V\MeanStack\Module 1\LoginForm-API\backend> node app.js
[dotenv@17.2.3] injecting env (2) from .env -- tip: ⚙ override existing env vars with { override: true }
Server running on port 3000
MongoDB Connected

```

Step 14: Verify that the login form is working correctly.

Step 15: Run the frontend application.

The terminal window shows the command `ng serve` being run and the output indicating the application bundle generation is complete, watch mode is enabled, and the local host URL is `http://localhost:4200/`.

```

● PS C:\Users\Admin\Documents\SEM-1V\MeanStack\Module 1\LoginForm-API> cd login
○ PS C:\Users\Admin\Documents\SEM-1V\MeanStack\Module 1\LoginForm-API\login> ng serve
Initial chunk files | Names | Raw size
main.js | main | 20.75 kB |
styles.css | styles | 95 bytes |

| Initial total | 20.84 kB

Application bundle generation complete. [1.564 seconds] - 2026-01-03T08:33:54.910Z

Watch mode enabled. Watching for file changes...
NOTE: Raw file sizes do not reflect development server per-request transformations.
→ Local: http://localhost:4200/
→ press h + enter to show help

```

LOGIN FORM

First Name

Last Name

Email

Password

Submit

Login successful!

Step 16: Check the database to confirm whether the user credentials are stored

MongoDB Compass - habiba/collegeDB.students

Connections Edit View Collection Help

Compass

- My Queries
- Data Modeling

CONNECTIONS (2)

- Search connections

- habiba
 - admin
 - collegeDB
 - students
 - config
 - local

habiba > collegeDB > students

Documents (1) Aggregations Schema Indexes (1) Validation

Type a query: { field: 'value' } or [Generate query](#)

ADD DATA EXPORT DATA UPDATE DELETE

```
_id: ObjectId('6958d8e4f59d992c6f8ae133')
firstName : "John"
lastName : "Smith"
email : "johnSmith41@gmail.com"
password : "1234567890"
__v : 0
```