

Assignment01_COMP3133_Screenshots

1. Server Running

```
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
[nodemon] starting `node src/index.js`
[dotenv@17.3.1] injecting env (6) from .env -- tip: 🔒 encrypt with Dotenvx: https://dotenvx.com
[nodemon] restarting due to changes...
[nodemon] starting `node src/index.js`
[dotenv@17.3.1] injecting env (6) from .env -- tip: 🔒 encrypt with Dotenvx: https://dotenvx.com
MongoDB connected successfully.
Server running at http://localhost:4000/graphql
[nodemon] restarting due to changes...
[nodemon] starting `node src/index.js`
[dotenv@17.3.1] injecting env (6) from .env -- tip: 🔒 prevent building .env in docker: https://dotenvx.com/prebuild
MongoDB connected successfully.
Server running at http://localhost:4000/graphql
[]
```

2. Signup - Success

The screenshot shows the Postman interface with a GraphQL query executed successfully. The query is a mutation named 'signup' that takes an input object with 'username', 'email', and 'password' fields. The response is a JSON object with a 'data' field containing a 'signup' object with 'success', 'message', and 'user' fields. The 'user' field contains the details of the newly created user.

Query:

```
1 mutation Signup($input: SignupInput!) {
2   signup(input: $input) {
3     success
4     message
5     user {
6       _id
7       username
8       email
9       created_at
10      updated_at
11    }
12  }
13 }
```

GraphQL Variables:

```
1 {
2   "input": {
3     "username": "Miranda2",
4     "email": "GustavoMiranda@example.com",
5     "password": "123456789"
6   }
7 }
8 }
```

Response (JSON):

```
1 {
2   "data": {
3     "signup": {
4       "success": true,
5       "message": "Signup successful.",
6       "user": {
7         "_id": "699dc7718973d961101e9962",
8         "username": "Miranda2",
9         "email": "GustavoMiranda@example.com",
10        "created_at": "171816017291",
11        "updated_at": "171816017291"
12      }
13     }
14   }
15 }
```

3. Login by Username - Success

The screenshot shows a Postman interface for a GraphQL endpoint at `http://localhost:4000/graphql`. The request is a POST method. The query is:

```
4 message
5 user {
6   _id
7   username
8   email
9   created_at
10  updated_at
11 }
12 }
13 }
14 }
```

The GraphQL variables are:

```
1 {
2   "username": "Miranda2",
3   "password": "123456789"
4 }
5 }
```

The response is a 200 OK status with a JSON body:

```
1 {
2   "data": {
3     "login": {
4       "success": true,
5       "message": "Login successful.",
6       "user": {
7         "_id": "699bc7718973d96118fe9962",
8         "username": "Miranda2",
9         "email": "gustavomiranda2@example.com",
10        "created_at": "1771816817291",
11        "updated_at": "1771816817291"
12      }
13     }
14   }
15 }
```

4. Login by Email - Success

The screenshot shows a Postman interface for a GraphQL endpoint at `http://localhost:4000/graphql`. The request is a POST method. The query is:

```
4 message
5 user {
6   _id
7   username
8   email
9   created_at
10  updated_at
11 }
12 }
13 }
14 }
```

The GraphQL variables are:

```
1 {
2   "email": "GustavoMiranda2@example.com",
3   "password": "123456789"
4 }
5 }
```

The response is a 200 OK status with a JSON body:

```
1 {
2   "data": {
3     "login": {
4       "success": true,
5       "message": "Login successful.",
6       "user": {
7         "_id": "699bc7718973d96118fe9962",
8         "username": "Miranda2",
9         "email": "gustavomiranda2@example.com",
10        "created_at": "1771816817291",
11        "updated_at": "1771816817291"
12      }
13     }
14   }
15 }
```

5. Add New Employee - Success

The screenshot shows the Postman interface with a workspace named "Gustavo Miranda's Workspace". The "API Network" tab is active, displaying a collection of API endpoints. The "POST Add" endpoint is selected, with the URL set to "http://localhost:4000/graphql". The request is a POST method. The "Body" tab is selected, showing a GraphQL query with variables. The query is:

```
15 query {
16   addNewEmployee(
17     input: {
18       first_name: "Samantha",
19       last_name: "Silva",
20       email: "employee_teste_01@example.com",
21       gender: "Female",
22       designation: "Software Engineer",
23       salary: 5000,
24       date_of_joining: "2020-02-20",
25       department: "IT",
26       employee_photo: "https://res.cloudinary.com/demo/image/upload/getting-started/shoes.jpg"
27     }
28   ) {
29     id
30     first_name
31     last_name
32     email
33     gender
34     designation
35     salary
36     date_of_joining
37     department
38     employee_photo
39     created_at
40     updated_at
41   }
42 }
```

The response is a 200 OK status, with a body containing a JSON object:

```
1 {
2   "data": {
3     "addNewEmployee": {
4       "success": true,
5       "message": "Employee added successfully.",
6       "employee": {
7         "id": "6990c8630973d96118fe996a",
8         "first_name": "Samantha",
9         "last_name": "Silva",
10        "email": "employee_teste_01@example.com",
11        "gender": "Female",
12        "designation": "Software Engineer",
13        "salary": 5000,
14        "date_of_joining": "1775456000000",
15        "department": "IT",
16        "employee_photo": "https://res.cloudinary.com/dc8dthcc/image/upload/v177517856/comp3133_assignment1_employees/pru26abk74pscdingyo.jpg",
17        "created_at": "177517869284",
18        "updated_at": "177517869284"
19      }
20    }
21  }
```

6. Get All Employees - Success

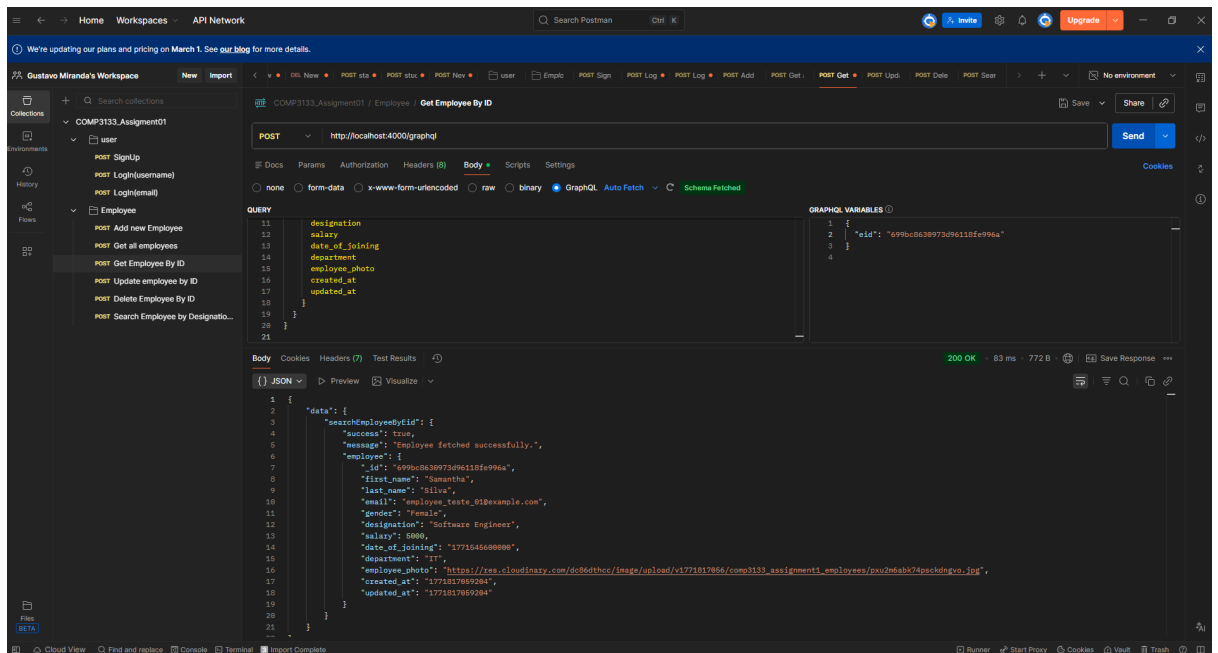
The screenshot shows the Postman interface with a workspace named "Gustavo Miranda's Workspace". The "API Network" tab is active, displaying a collection of API endpoints. The "POST Get" endpoint is selected, with the URL set to "http://localhost:4000/graphql". The request is a POST method. The "Body" tab is selected, showing a GraphQL query with variables. The query is:

```
7 query {
8   getAllEmployees {
9     id
10    first_name
11    last_name
12    email
13    gender
14    designation
15    salary
16    date_of_joining
17    department
18    employee_photo
19    created_at
20    updated_at
21  }
22 }
```

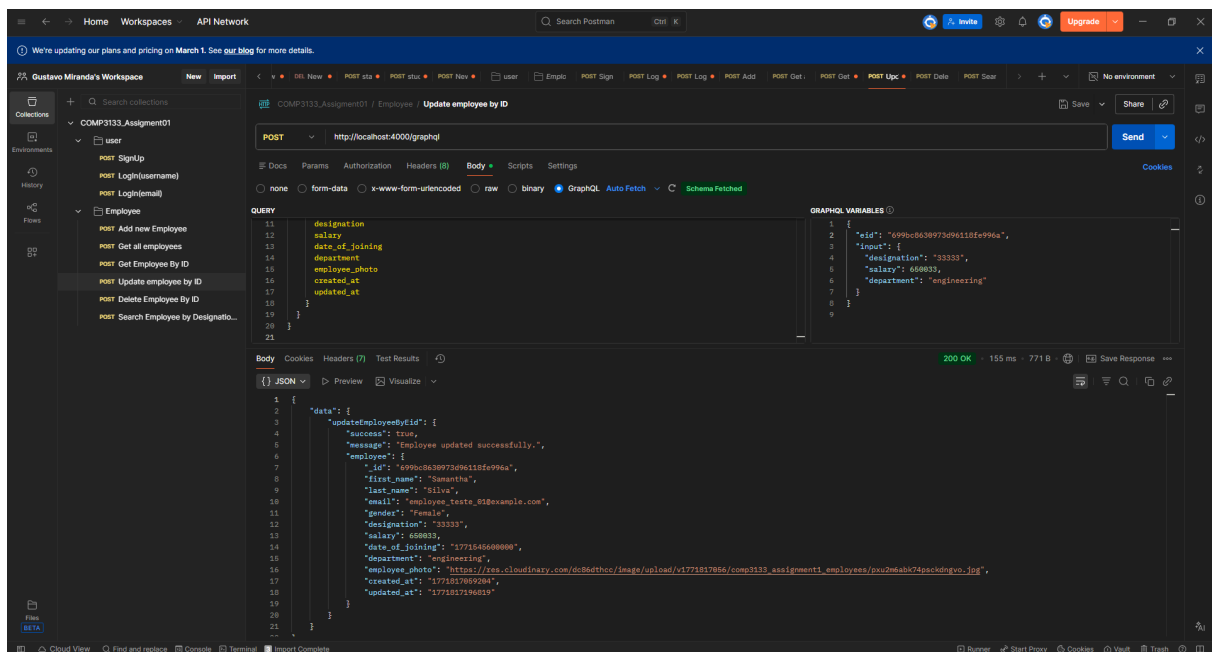
The response is a 200 OK status, with a body containing a JSON object:

```
1 {
2   "data": {
3     "getAllEmployees": [
4       {
5         "id": "6990c8630973d96118fe996a",
6         "first_name": "Samantha",
7         "last_name": "Silva",
8         "email": "employee_teste_01@example.com",
9         "gender": "Female",
10        "designation": "Software Engineer",
11        "salary": 5000,
12        "date_of_joining": "1775456000000",
13        "department": "IT",
14        "employee_photo": "https://res.cloudinary.com/dc8dthcc/image/upload/v177517856/comp3133_assignment1_employees/pru26abk74pscdingyo.jpg",
15        "created_at": "177517869284",
16        "updated_at": "177517869284"
17      }
18    ]
19  }
```

7. Search Employee by ID - Success



8. Update Employee by ID - Success



9. Search by Designation or Department - Success

The screenshot shows the Postman interface with a workspace named "Gustavo Miranda's Workspace". The API Network tab is active, showing a collection named "COMP3133_Assignment01". The "Employee" folder is expanded, and the "Search Employee by Designation or Department" endpoint is selected. The endpoint is a POST request to "http://localhost:4000/graphql". The request body is a GraphQL query with variables. The query is:

```
1 query {
2   searchEmployeeByDesignationOrDepartment(
3     designation: "33333",
4     department: "engineering"
5   ) {
6     id
7     first_name
8     last_name
9     email
10    gender
11    designation
12    salary
13    date_of_joining
14    employee_photo
15    created_at
16    updated_at
17  }
18 }
```

The GraphQL variables are:

```
1 {
2   designation: "33333",
3   department: "engineering"
4 }
```

The response is a 200 OK status with a response time of 79 ms and a size of 722 B. The response body is a JSON object:

```
1 {
2   "data": {
3     "searchEmployeeByDesignationOrDepartment": [
4       {
5         "id": "699bc638973d96110fe99ea",
6         "first_name": "Samantha",
7         "last_name": "Silva",
8         "email": "employee_teste_01@example.com",
9         "gender": "Female",
10        "designation": "33333",
11        "salary": 698833,
12        "date_of_joining": "177154668000",
13        "department": "engineering",
14        "employee_photo": "https://res.cloudinary.com/dc8dthccc/image/upload/v1771817856/comp3133_assignment1_employees/pzu2h6abk74pscdngvq.jpg",
15        "created_at": "1771817856000",
16        "updated_at": "1771817856000"
17      }
18     ]
19   }
20 }
```

10. Delete Employee by ID - Success

The screenshot shows the Postman interface with a workspace named "Gustavo Miranda's Workspace". The API Network tab is active, showing a collection named "COMP3133_Assignment01". The "Employee" folder is expanded, and the "Delete Employee by ID" endpoint is selected. The endpoint is a POST request to "http://localhost:4000/graphql". The request body is a GraphQL query with variables. The query is:

```
1 mutation DeleteById($id: ID!) {
2   deleteEmployeeById(id: $id) {
3     success
4     message
5   }
6 }
```

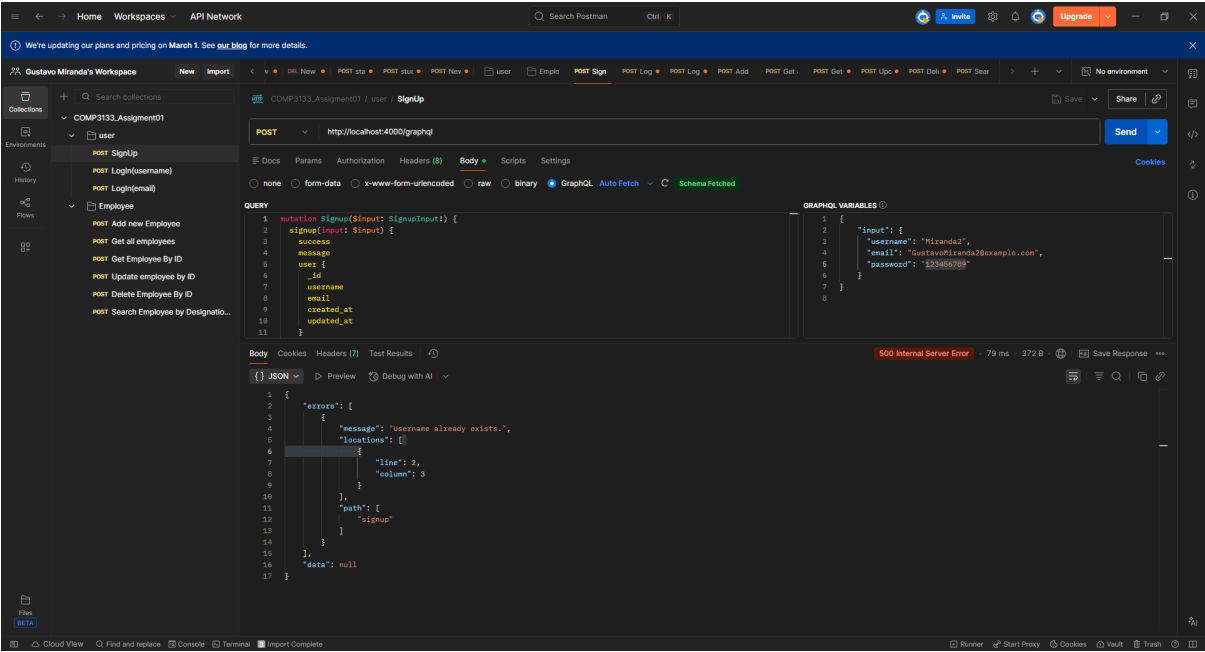
The GraphQL variables are:

```
1 {
2   id: "699bc638973d96110fe99ea"
3 }
```

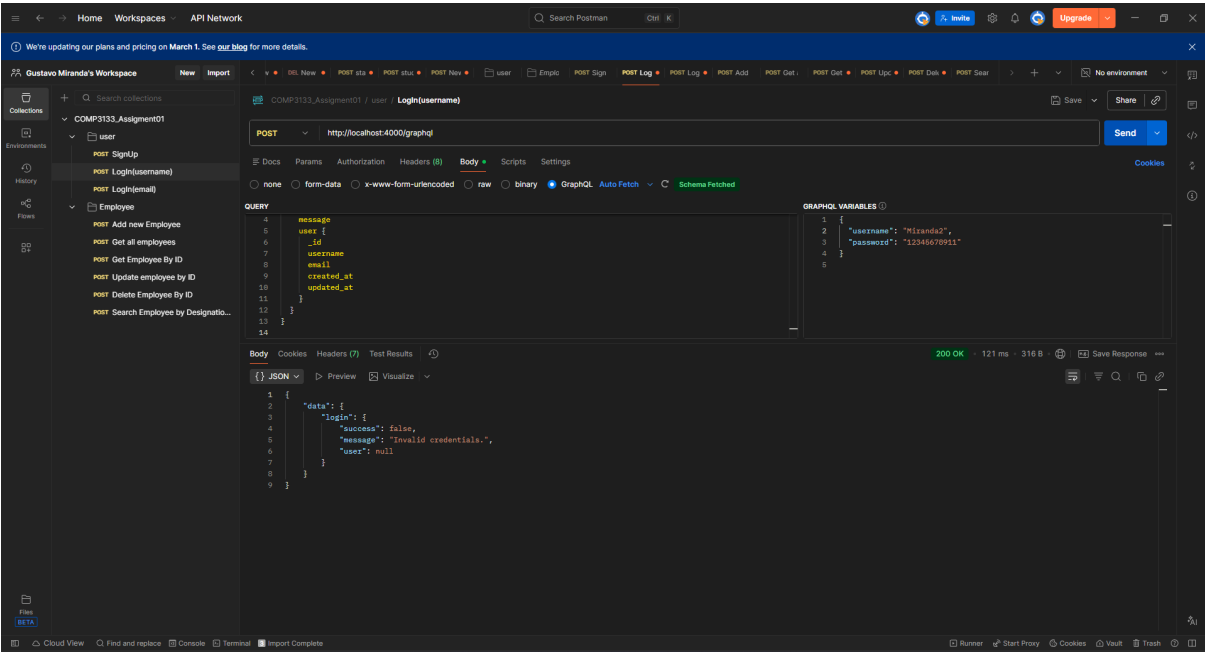
The response is a 200 OK status with a response time of 79 ms and a size of 327 B. The response body is a JSON object:

```
1 {
2   "data": {
3     "deleteEmployeeById": {
4       "success": true,
5       "message": "Employee deleted successfully."
6     }
7   }
8 }
```

11. Signup - Duplicate Username Error



12. Login - Wrong Password Error



13. Add Employee - Salary Validation Error

The screenshot shows the Postman interface for a GraphQL API. The collection is 'COMP3133_Assignment01' and the environment is 'user'. The selected endpoint is 'POST http://localhost:4000/graphql'. The request body is a GraphQL mutation to add a new employee with the following variables:

```
GRAPHQL VARIABLES {
  1 "input": {
  2   "first_name": "Sa32",
  3   "last_name": "Silva33",
  4   "email": "employee_teste_012@example.com",
  5   "gender": "Female",
  6   "designation": "Software Engineer",
  7   "salary": 900,
  8   "date_of_joining": "2020-02-20",
  9   "department": "IT",
 10   "employee_photo": "https://res.cloudinary.com/demo/image/upload/getting-started/shoes.jpg"
 11 }
}
```

The response is a 500 Internal Server Error (4 ms, 392 B). The JSON body shows an error message: "Salary must be a number and >= 1000.".

```
{
  "errors": [
    {
      "message": "Salary must be a number and >= 1000.",
      "locations": [
        {
          "line": 2,
          "column": 3
        }
      ],
      "path": [
        "addNewEmployee"
      ]
    }
  ],
  "data": null
}
```

14. Add Employee - Gender Validation Error

The screenshot shows the Postman interface for a GraphQL API. The collection is 'COMP3133_Assignment01' and the environment is 'user'. The selected endpoint is 'POST http://localhost:4000/graphql'. The request body is a GraphQL mutation to add a new employee with the following variables:

```
GRAPHQL VARIABLES {
  1 "input": {
  2   "first_name": "Sa32",
  3   "last_name": "Silva33",
  4   "email": "employee_teste_012@example.com",
  5   "gender": "1",
  6   "designation": "Software Engineer",
  7   "salary": 1100,
  8   "date_of_joining": "2020-02-20",
  9   "department": "IT",
 10   "employee_photo": "https://res.cloudinary.com/demo/image/upload/getting-started/shoes.jpg"
 11 }
}
```

The response is a 500 Internal Server Error (4 ms, 393 B). The JSON body shows an error message: "Gender must be Male, Female or Other.".

```
{
  "errors": [
    {
      "message": "Gender must be Male, Female or Other.",
      "locations": [
        {
          "line": 2,
          "column": 3
        }
      ],
      "path": [
        "addNewEmployee"
      ]
    }
  ],
  "data": null
}
```

15. Search Employee by ID - Invalid ID Error

The screenshot shows the Postman interface with a workspace named 'Gustavo Miranda's Workspace'. The left sidebar lists collections, including 'COMP3133_Assignment01' and 'user'. The 'Employee' collection is expanded, showing a list of endpoints. The 'POST Get Employee by ID' endpoint is selected. The main panel shows the query details for a POST request to 'http://localhost:4000/graphql'. The query is a GraphQL query to search for an employee by ID. The response is a 500 Internal Server Error, indicating an invalid employee ID.

Query:

```
11  query {
12    designation
13    salary
14    date_of_joining
15    department
16    employee_photo
17    created_at
18    updated_at
19  }
20 }
21 }
```

GraphQL Variables:

```
1 {
2   "id": "21"
3 }
4 }
```

Response (JSON):

```
1 {
2   "errors": [
3     {
4       "message": "Invalid employee id.",
5       "locations": [
6         {
7           "line": 2,
8           "column": 3
9         }
10      ],
11      "path": [
12        "searchEmployeeById"
13      ]
14    }
15  ],
16  "data": null
17 }
```

16. Update Employee by ID - Not Found Error

The screenshot shows the Postman interface with a workspace named 'Gustavo Miranda's Workspace'. The left sidebar lists collections, including 'COMP3133_Assignment01' and 'user'. The 'Employee' collection is expanded, showing a list of endpoints. The 'POST Update employee by ID' endpoint is selected. The main panel shows the query details for a POST request to 'http://localhost:4000/graphql'. The query is a GraphQL query to update an employee by ID. The response is a 500 Internal Server Error, indicating an invalid employee ID.

Query:

```
11  query {
12    designation
13    salary
14    date_of_joining
15    department
16    employee_photo
17    created_at
18    updated_at
19  }
20 }
21 }
```

GraphQL Variables:

```
1 {
2   "id": "6990c8638973d96310fe996",
3   "input": {
4     "designation": "33333",
5     "salary": 650033,
6     "department": "engineering"
7   }
8 }
9 }
```

Response (JSON):

```
1 {
2   "errors": [
3     {
4       "message": "Invalid employee id.",
5       "locations": [
6         {
7           "line": 2,
8           "column": 3
9         }
10      ],
11      "path": [
12        "updateEmployeeById"
13      ]
14    }
15  ],
16  "data": null
17 }
```

17. Delete Employee by ID - Not Found Error

The screenshot shows the Postman interface for a workspace named 'Gustavo Miranda's Workspace'. The selected collection is 'COMP3133_Assignment01' and the environment is 'user'. The request is a POST to 'http://localhost:4000/graphql' with the following GraphQL query:

```
1 mutation DeleteEmployeeById($id: ID!) {
2   deleteEmployeeById(id: $id) {
3     success
4     message
5   }
6 }
7
```

The GraphQL variables are:

```
1 {
2   "id": "6996c86309773d96118fe99"
3 }
4
```

The response is a 500 Internal Server Error (3 ms, 381 B). The JSON body shows the error:

```
1 {
2   "errors": [
3     {
4       "message": "Invalid employee id.",
5       "locations": [
6         {
7           "line": 2,
8           "column": 3
9         }
10      ],
11      "path": [
12        "deleteEmployeeById"
13      ]
14    }
15  ],
16  "data": null
17 }
```

18. Search by Designation or Department - Missing Filters Error

The screenshot shows the Postman interface for the same workspace. The selected collection is 'COMP3133_Assignment01' and the environment is 'user'. The request is a POST to 'http://localhost:4000/graphql' with the following GraphQL query:

```
1 query {
2   searchEmployeeByDesignationOrDepartment {
3     gender
4     designation
5     salary
6     date_of_joining
7     department
8     employee_photo
9     created_at
10    updated_at
11  }
12 }
13
```

The GraphQL variables are:

```
1 {
2   "designation": "",
3   "department": ""
4 }
5
```

The response is a 500 Internal Server Error (4 ms, 415 B). The JSON body shows the error:

```
1 {
2   "errors": [
3     {
4       "message": "Provide designation or department.",
5       "locations": [
6         {
7           "line": 2,
8           "column": 3
9         }
10      ],
11      "path": [
12        "searchEmployeeByDesignationOrDepartment"
13      ]
14    }
15  ],
16  "data": null
17 }
```

19. MongoDB Atlas - Database and Collections

The screenshot shows the MongoDB Atlas web interface. The left sidebar contains the 'Data Explorer' with a tree view of clusters. The main panel displays the 'Users' collection. The document list shows two documents with fields like `_id`, `username`, `email`, `password`, `created_at`, and `updated_at`. The interface includes navigation tabs for 'Documents', 'Aggregations', 'Schema', 'Indexes', and 'Validation'. A search bar and query editor are at the top of the main panel.

System Status: All Good
©2024 MongoDB, Inc. Status Terms Privacy Atlas Blog Contact Sales

The screenshot shows the MongoDB Atlas web interface with the 'Employee' collection selected. The document list displays a single document with fields including `_id`, `first_name`, `last_name`, `email`, `gender`, `designation`, `salary`, `date_of_joining`, `department`, `employee_photo`, `created_at`, and `updated_at`. The interface features the same navigation and search components as the previous screenshot.

System Status: All Good
©2024 MongoDB, Inc. Status Terms Privacy Atlas Blog Contact Sales