### ****Explanation of the Endpoints and Error Handling****

### ****1:Endpoints Explanation****

**List All Users**

**Route**: /users

**Description**: Reads and returns all users from the users.json file.

**Implementation**:

Uses the fs.readFile method to read the JSON file containing user data.

Parses and sends the entire data object as the response.

**Example Request**: GET http://localhost:5000/users

**Example Response**

{

"user1": {

"name": "Alice",

"password": "pass123",

"profession": "developer",

"id": 1

},

"user2": {

"name": "Bob",

"password": "pass456",

"profession": "designer",

"id": 2

},

"user3": {

"name": "Charlie",

"password": "pass789",

"profession": "manager",

"id": 3

}

}

**Get User by ID**

**Route**: /users/:id

**Description**: Returns the details of a specific user based on their id.

**Implementation**:

Extracts the id from the route parameter.

Filters users from the JSON data to find the matching ID.

**Example Request**: GET <http://localhost:5000/users/1>

**Example Response**:

{

"name": "Alice",

"password": "pass123",

"profession": "developer",

"id": 1

}

**Get Users by Profession**

**Route**: /users/profession/:profession

**Description**: Returns all users with a specific profession.

**Implementation**:

Extracts the profession parameter from the route.

Filters users based on the case-insensitive match of their profession.

**Example Request**: GET <http://localhost:5000/users/profession/developer>

**Example Response**:

[

{

"name": "Alice",

"password": "pass123",

"profession": "developer",

"id": 1

}

]

**Get User by Name**

**Route**: /users/name/:name

**Description**: Returns the user with the given name.

**Implementation**:

Extracts the name parameter from the route.

Filters users for a case-insensitive name match.

**Example Request**: GET <http://localhost:5000/users/name/Alice>

Example Response:

{

"name": "Alice",

"password": "pass123",

"profession": "developer",

"id": 1}

#### ****Error Handling****

#### ****File Read Errors****:

**Scenario**: If the users.json file cannot be read (e.g., missing or corrupted).

**Implementation**:

Checks for fs.readFile errors and sends a 500 Internal Server Error with the message "Error reading the users file."

**Invalid User ID**:

**Scenario**: The requested id does not exist in the data.

**Implementation**:

Checks if the filtered result for id is null or undefined.

Returns a 404 Not Found with the message "User not found."

**Invalid Profession**:

**Scenario**: No users match the requested profession.

**Implementation**:

Checks if the filtered results array is empty.

Returns a 404 Not Found with the message "No users found with the given profession."

**Invalid Name**:

**Scenario**: No user matches the requested name.

**Implementation**:

Checks if the filtered result for name is null or undefined.

Returns a 404 Not Found with the message "User not found."

#### ****Overall****

The API is designed with clear endpoints to retrieve user data from the users.json file. Robust error handling ensures proper feedback for invalid requests or system errors. The implementation leverages the fs module for file operations and filters data dynamically based on request parameters​(server)​(user).