MERN Magic Query Builder Challenge

With the following exercise we are meant to assess your ability to:

- Analyze and understand complex problems and be able to break them down into smaller chunks which are easier to solve
- Write clean and readable code easy to review
- Apply design principles in favor of reusability and scalability
- Think out-of-the-box to come up with smart and creative solutions
- Work on schedule, delivering quick high-quality products

There is no time limit for completing this exercise, but once you start, we suggest not to spend more than 4 days

Do not stress and try to find the right balance that works for you.

Objective: Build a full-stack application using the MERN stack (MongoDB, Express, React, and Node.js) that interprets a magic query language and displays the results.

Target Skills:

* Front-end development with React:
    * Building UI for creating and managing queries.
    * Rendering generated JSON based on input query.
    * Implementing optional features like query history and visual editor.
* Back-end development with Node.js and Express:
    * Implementing API endpoint for receiving query and returning JSON.
    * Building logic to interpret magic query and build JSON.
    * Handling nested objects, arrays, `__match__`, `__eq__`, `__gt__`, `__or__`, and `__and__` keywords.
    * Integrating with MongoDB for data storage and retrieval.
    * Implementing user login and registration for data access control.
* RESTful API design and implementation:
    * Designing and developing RESTful API for data access and manipulation.
    * Ensuring proper use of HTTP methods and response codes.
* Code quality and testing:
    * Writing clean, well-structured, and maintainable code.
    * Implementing unit tests for key functionalities.
* Deployment:
    * Deploying the application to a cloud platform like Heroku or AWS.

Examples:

Example 1: Nested Arrays and `__match__` keyword

Input Query:

```
users.[].orders.[].products.[].name = "phone" __AND__ price > 1000
```

Generated JSON Object:

json

```
{
  "__query__": {
    "users": {
      "__match__": {
```

```
        "orders": {
          "__match__": {
            "products": {
              "__match__": {
                "name": "phone",
                "price": {
                  "__gt__": 1000
                }
              }
            }
          }
        }
      }
    }
  }
}
```

Database Operation:

Find all users whose orders contain at least one product where the name is "phone" and the price is greater than 1000.

Example 2: Multiple Conditions and `__eq__`, `__or__`, and `__and__` keywords

Input Query:

```
(category = "electronics" __OR__ category = "clothing") __AND__ (brand = "Samsung" __OR__ brand = "Apple")
```

Generated JSON Object:

```
{
  "__query__": {
    "__or__": [
      {
        "category": {
          "__eq__": "electronics"
        }
      },
      {
        "category": {
          "__eq__": "clothing"
        }
      }
    ]
  },
  "__and__": [
    {
      "brand": {
        "__eq__": "Samsung"
      }
    },
    {
      "brand": {
        "__eq__": "Apple"
      }
    }
  ]
}
```

Database Operation:

Find all products where either the category is "electronics" or "clothing" and the brand is either "Samsung" or "Apple".

Additional Example 3: Filtering with Dates

Input Query:

```
orders.created_at > "2023-10-26" __AND__ orders.status = "shipped"
```

Generated JSON Object:

```
{
  "__query__": {
    "orders": {
      "__match__": {
        "created_at": {
          "__gt__": "2023-10-26"
        },
        "status": {
          "__eq__": "shipped"
        }
      }
    }
  }
}
```

Database Operation:

Find all orders created after October 26th, 2023, that have a status of "shipped".

Bonus Features:

* Query history: Track past queries for easy reuse.
* Visual editor: Build queries using drag-and-drop and pre-defined components.
* Data visualization: Integrate with Chart.js or D3.js for visual representation of query results.

Evaluation Criteria:

* Functionality: Completeness of required functionalities and handling of all scenarios.
* User Interface: User-friendliness, intuitiveness, and responsiveness.
* Code Quality: Cleanliness, organization, ease of understanding, and adherence to best practices.

* Security: Correct implementation of user authentication and proper