

Design and Implementation of a Data Filtering and Sorting Interface in React

A Technical Report submitted to the Department of Computer Science

Presented to the Faculty of the School of Engineering and Applied Science
University of Virginia • Charlottesville, Virginia

In Partial Fulfillment of the Requirements for the Degree
Bachelor of Science, School of Engineering

Calvin Kuo

Spring 2023

On my honor as a University Student, I have neither given nor received unauthorized aid on this assignment as defined by the Honor Guidelines for Thesis-Related Assignments

Briana Morrison, Department of Computer Science

Design and Implementation of a Data Filtering and Sorting Interface in React

CS 4991 Capstone Report, 2022

Calvin Kuo

Computer Science

The University of Virginia

School of Engineering and Applied Science

Charlottesville, Virginia, USA

clk3sx@virginia.edu

Abstract

The Neonatal Antibiotic Stewardship at the University of Virginia provides a web form to help clinicians prescribe the appropriate dosage and type of antibiotics. Although the system collects the inputted data into a MongoDB database, there was no user-friendly way to access the data. Using an agile development methodology, my internship team designed and implemented a user interface with the React framework to display this data. This enabled the user to specify an arbitrary number of filters and sorts, allowing a non-technical user to easily view the desired subset of the data. In addition, we also designed the interface to allow later extension and customization with additional features and visualizations of the relevant data.

1. Introduction

Antibiotics have saved millions of lives since the discovery of the first antibiotic, penicillin, almost a century ago. They work by killing bacteria, which prevents infections from progressing and causing further illness. However, in recent decades, biologists have found an increasing number of strains of bacteria have evolved so that they can survive antibiotic treatment. This phenomenon is known as antibiotic resistance.

To prevent this development from undoing the past century of medical advances,

clinicians need to use a treatment that meets two criteria. First, it should be a narrow-spectrum antibiotic, which affects only the type of bacteria at issue. This minimizes the collateral damage caused by killing good bacteria, such as the ones that aid digestion in the gut. Second, the dosing regimen should ensure that all of the bacteria of interest are killed. Otherwise, the bacteria which survive treatment may continue to multiply, causing an infection resistant to the previous treatment used.

2. Related Works

The concept of a relational database in the context of a computer system dates back to the late 1960s (Codd, 1970). The concept of a graphical interface based on entities was introduced in 1980, and a user interface that could also express generic queries was introduced in 1982 (Cattell, 1980; Wong and Kuo, 1982).

Spreadsheet programs such as VisiCalc, Lotus 1-2-3, and Microsoft Excel have their origins in the late 1970s and 1980s (Grad, 2007). Software like these brought the concept of organizing data on computers in a tabular format to the masses, and as such, have had a significant influence on how users expect to interact with data. Microsoft Excel, Numbers, and Google Sheets are the major players in the spreadsheet software market today.

Dzafic et al. (2013) also considered the user interface design implications of an object-oriented database. They also discuss how this approach affects the development life cycle and continued maintenance.

3. Process Design

Under the supervision of David Kaufman, MD, of the University of Virginia School of Medicine, the Neonatal Antibiotic Stewardship and the University of Virginia Development Hub developed the Clinical Decision Support Tool (Figure 1), a web form which helps neonatal clinicians determine which course of antibiotics would be the most appropriate based on a number of factors.

3.1 Task Description

For the tool to provide the most useful information, all inputted queries are saved into a database. However, at the beginning of my internship, there was no way to access the data without logging into the database backend and viewing the raw inputs. My internship team was tasked with designing and implementing a web application that would allow a non-technical user to view this information.

3.2 System Architecture

The primary technologies used in both the Clinical Decision Support Tool and the Data Trends app was the React framework and the MongoDB data platform. React is an open-source frontend JavaScript library, originally developed by Facebook, that allows developers to build user interfaces out of reusable components. MongoDB is a JavaScript-based document-oriented web platform that served as the database backend for this project. JavaScript is a programming language that runs in browsers and is what makes many websites interactive.

As we were designing the Data Trends app from scratch, the design challenges we faced

The screenshot displays a web form titled "Age and Weight" with the following sections:

- Age and Weight:** Gestational Age (28 weeks, 0 days), Postnatal Age (at time of culture sent) (3 days), Birth Weight (444 grams), Current Weight (at time of form completion) (333 grams).
- Early-Onset (EOS) or Late-Onset (LOS) Sepsis:** Radio buttons for EOS (less than 72 hours after birth) and LOS (72 or more hours after birth).
- Pathogen Isolated:** Radio buttons for Yes and No. A dropdown menu is set to "E. Coli". A note says "(You enter Gram stain or specific species)".
- Susceptibility Results:** Radio buttons for Pending and Known.
- Site of Infection:** Checkboxes for None OR Culture Results Pending, Blood, Urine, CSF (checked), Peritoneal, and Skin with Cellulite. A note says "(check all that apply)".
- Abdominal Involvement Present?:** Radio buttons for Yes and No.
- Footer:** A checkbox for "I have read and accepted the terms and conditions" and "Submit" and "Clear" buttons.

Figure 1: The web form portion of the Clinical Decision Support Tool. It consists of nine questions, with the provided answers being used to query the backend database.

included implementing the functionality of filters, sorts, graphs, and results in an easy-to-use manner so that the client would be able to view and analyze the collected data. In order for the Data Trends app to better integrate with the Clinical Decision Support

Tool, we also used React and MongoDB to build it.

3.3 Key Components

The Data Trends app consists of four main functionalities: filtering, sorting, displaying results, and importing/exporting.

3.3.1 Filters Component

The filters component allows the user to specify an arbitrary number of filters for their query (Figure 2). Each row of the component describes a single filter. It consists of three parts: a dropdown to select a column of the database, a dropdown to select a condition to apply to that column, and any number of arguments to that condition. Because the attribute values can be one of several data types, the available options change to only show the ones that are relevant (Figure 3). For numerical values such as Birth Weight, the available conditions include $>$, $<$, and *is between*. For text values such as Site of Infection, the available conditions include regular expressions.



Figure 2: The filters portion of the Data Trends app.



Figure 3: The dropdown menus from the inputs shown in Figure 2 when expanded.



Figure 4: Dictionaries specifying the behavior of a column (top) and condition (bottom).

To implement this, the React component stores the current list of filters in a state as an array of objects. In these, the values for the column and condition are used to index into dictionaries indicating how the form fields should be displayed (Figure 4). To allow for behaviors to be changed without having to make modifications to the SearchFilter component, they are described in a separate file rather than implemented within the rendering logic.

3.3.2 Sorting Component

The sorting interface is largely a pared-down version of the filters interface (Figure 5). The main difference is the addition of buttons to move up and down individual sorts. This is useful because the order the sorts are applied in can change the order of

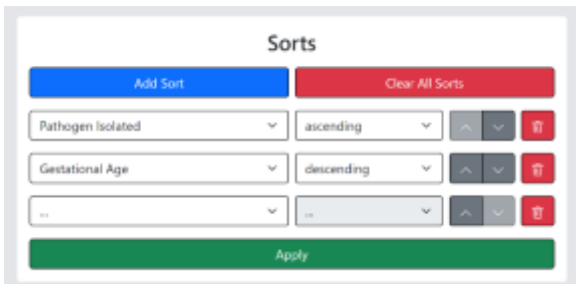


Figure 5: The sorting portion of the Data Trends app.

the results. This issue does not occur with filters, since AND is a commutative operator.

3.3.3 Results Component

When the Apply button is clicked, the filters and sorts are converted into a format that can be understood by MongoDB, and the query is sent to the database. Once the app receives a response, it passes the data into the Visualizations component, which renders various graphs and charts, and into the View component, which renders a table of the results (Figure 6).

3.3.4 Import/Export

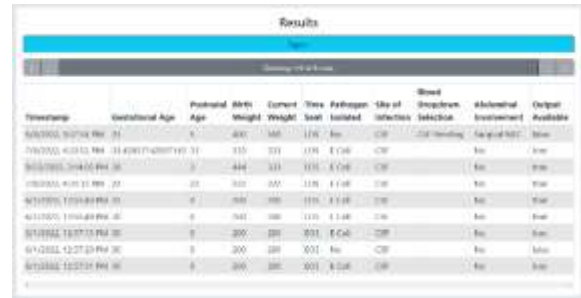
Queries can be saved to and loaded from a file on disk using the JSON format. It includes the current state of the SearchFilter and SearchSort components, and allows the form to be repopulated in the exact same state as when it was saved. This is the same format that is saved to local storage to remember queries between sessions and when navigating using the back/forward buttons.

4. Results

The Data Trends app was approved by the client and succeeded in allowing non-technical users to view the data inputted into the database. Additionally, the graphs will allow the client to see at a glance what kinds of queries users are making and where to improve the recommendations that the tool provides.

One of the limitations of the filters interface is that it does not allow users to create queries of arbitrary complexity. For example, any queries that require the Boolean OR operator, such as “(gestational age < 25) OR (gestational age > 30),” cannot be entered. Only queries of the form “*a* AND *b* AND *c* AND ...” can be entered.

To allow queries that use OR and NOR, I implemented a dropdown that would let the



Timestamp	Gestational Age	Estimated Weight	Current Weight	Fetal Growth	Blood Pressure	Abnormal	Output
2017-01-01 10:00 AM	30	300	300	100	120	No	low
2017-01-01 11:00 AM	31	310	310	100	120	No	low
2017-01-01 12:00 PM	32	320	320	100	120	No	low
2017-01-01 13:00 PM	33	330	330	100	120	No	low
2017-01-01 14:00 PM	34	340	340	100	120	No	low
2017-01-01 15:00 PM	35	350	350	100	120	No	low
2017-01-01 16:00 PM	36	360	360	100	120	No	low
2017-01-01 17:00 PM	37	370	370	100	120	No	low
2017-01-01 18:00 PM	38	380	380	100	120	No	low
2017-01-01 19:00 PM	39	390	390	100	120	No	low
2017-01-01 20:00 PM	40	400	400	100	120	No	low

Figure 6: The results portion of the Data Trends app.

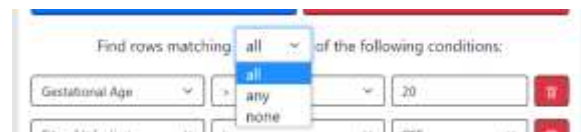


Figure 7: A rejected feature that would have allowed users to specify how the conditions should be combined.

user choose how they would like the conditions they specified to be combined (Figure 7). However, one of my supervisors rejected this idea as being unnecessarily confusing for most use cases. Because of this, I did not spend any time working on an interface for nested queries.

Currently, the calculations required for the graphs and the pagination of the results is all handled in the browser on the client side. This allows changing the number of items per page and jumping between pages to occur without the latency from making an additional HTTP request. This may not work with larger data sets, however.

5. Conclusion

The development of the Data Trends app shows that React is a good framework for producing a dynamic application. The three of us had minimal exposure to React before this internship, but learned it fairly quickly. By the time we started developing the Data Trends app, we had only one month of experience with developing with React. Over the course of two weeks, we managed

to create a minimum viable product with all of the requested features, including filtering and sorting, using an agile development methodology. Ultimately, it will help the Neonatal Antibiotic Stewardship gather more data about how to best treat newborn babies with antibiotics.

6. Future Work

The development phase of the application is now complete. However, it is likely that additional support work will be done on the existing codebase until the project is completely rewritten or abandoned.

Although the application works for small datasets, we did not test it on larger datasets. These might take up a significant amount of memory or bandwidth, making it impossible for the client to process the data efficiently. In this case, some functionality may need to be moved to the server side instead.

Additionally, very few tests were written for the application. If the application were to be used in an environment with more stringent code standards, these would need to be written to ensure the program's correctness.

7. Acknowledgments

I would like to thank Nadim El-Jaroudi, Morgan Hale, and William Bigger from the University of Virginia Development Hub for their valuable guidance, as well as my fellow interns Scott Hong and Steven Song for their assistance and insight.

References

E. F. Codd. 1970. A relational model of data for large shared data banks. *Commun. ACM* 13, 6 (June 1970), 377–387. <https://doi.org/10.1145/362384.362685>

R. G. G. Cattell. 1980. An entity-based database user interface. In *Proceedings of the 1980 ACM SIGMOD international*

conference on Management of data (SIGMOD '80). Association for Computing Machinery, New York, NY, USA, 144–150. <https://doi.org/10.1145/582250.582273>

I. Dzafic, J. Sofu, E. Halilovic, N. Lecek and M. Music. 2013. Object-oriented database and user interface design. *Eurocon 2013*, 558-563. <https://doi.org/10.1109/EUROCON.2013.6625036>.

B. Grad. 2007. The Creation and the Demise of VisiCalc. In *IEEE Annals of the History of Computing*, 29, 3 (July-Sept. 2007), 20-31. <https://doi.org/10.1109/MAHC.2007.4338439>.

Harry K. T. Wong and Ivy Kuo. 1982. GUIDE: Graphical User Interface for Database Exploration. In *Proceedings of the 8th International Conference on Very Large Data Bases (VLDB '82)*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 22–32. <https://www.vldb.org/conf/1982/P022.PDF>