Faculty Code: EAR
Project Number: 1803

Adverse Reaction Reports Analysis Tool

A Major Qualifying Project Report:
Submitted to the Faculty of the
WORCESTER POLYTECHNIC INSTITUTE

WPI

In partial fulfillment of the requirements for the
Degree of Bachelor of Science

Submitted by:

Derek Murphy
Oliver Spring
Cory Tapply
Daniel Yun

Date: March 22, 2018

Approved by:

Professor Elke A. Rundensteiner

Acknowledgements

We would like to express gratitude and acknowledge the following individuals for all their help, support, and contributions to this project:

- WPI graduate students Tabassum Kakar and Xiao Qin who performed all the preliminary research for this project as well as helped guide us through building the application through the entire project by meeting with the team weekly and continuing to research what features the FDA want the most.
- Professor Elke Rundensteiner from WPI for her assistance and guidance throughout the entire project.
- We would like to acknowledge the FDA for providing the publicly available data used for this project.

Abstract

The purpose of this project is to provide a working alternative to the FDA Drug Safety Evaluator's current medical error analysis workflow. By prioritizing a visualization-focused application, investigators are able to easily identify trends and important reports. Using treemap visualizations, bar charts, and area charts, an investigator is able to sift through medical error reports by patient demographic, report statistics, and date, allowing specific reports to be found easily. Specific reports have all of their metadata displayed in a tabular view and can be interacted with to show the report's narrative. The report narrative can be annotated with highlights to point out significant information. The application allows for the evaluators to mark and store significant reports in cases that provide further statistical analysis on the annotated reports stored inside them. The application is incomplete but has the functionality for many use-cases and will continue to be developed in the future.

Table of Contents

1. Introduction

Pharmacovigilance is the practice of monitoring the effects of already-approved drugs to identify and evaluate previously undescribed adverse reactions [1]. The Food and Drug Administration's (FDA) Division of Pharmacovigilance (DPV) documents and analyzes unforeseen adverse reactions that may arise from drug usage. Since new drugs are constantly released, it is important to carefully monitor and analyze any adverse reactions they may cause to gain a clear understanding of what may have caused them.

The FDA currently has a database it uses to store adverse reaction data called the FDA Adverse Event Reporting System (FAERS). It receives approximately one million reports each year and is used to detect potential drug reaction patterns. The FAERS database currently holds 14 million records of reactions dating back to 1968, eight million of which classify as serious and 1.4 million that resulted in death [2]. The FDA DPV prioritizes reports based on many different criteria of the adverse events such as reaction type, cause, and event outcome. Finding the most severe reports to quickly determine the cause of an adverse reaction is a complex process. Adverse events can be unforeseeable, but they only account for one type of report that the FAERS receives.

The other type of report is a Medical Error report. Medical errors, unlike adverse events, are avoidable and deal with side effects, for example, that could arise due to the patients receiving the wrong medication. The FDA's Division of Medical Error Prevention and Analysis (DMEPA) is responsible for handling these reports. Several factors can lead to medication errors. Chief among them are drug name confusion and packaging-labeling similarities. For instance,

one could easily confuse the drugs Durezol, a prescription eye drop medication, and Durasal, a prescription wart remover, because of their similar names and similarly shaped containers [3]. Medication errors that occur while prescribing, repackaging, dispensing, administering or monitoring are all kept and carefully examined.

1.1 The Problem

Each report contains potentially important information and must be individually analyzed through drug name, severity, patient outcome, and keywords that may be found in the report text in order to detect trends. Medical errors can has serious consequences, including injury or death. It is important that employees are able to detect the causation of widespread medical errors as soon as possible.

With the importance of this data in mind, the development of a tool that allows the FDA DPV to easily visualize trends by applying filters based on date, demographics, and medication types and errors is of utmost importance. Once the FDA DPV has come to the conclusion that a report is part of a trend, they can then add the report to that trend's case. This provides a more effective way of searching through reports based on key attributes in large quantities to narrow down trends and come to conclusions.

1.2 Goal of the MQP

The goal of this MQP is to build a tool for the FDA DPV that allows its Drug Safety Evaluators to analyze relevant data from the FAERS database quickly. For this a data analysis and visualization tool must be created that:

- Facilitates the analysis of medication errors and adverse effects in pharmacovigilance
- Can display various representations of the data contained in the medication error database
- Can view and annotate narratives of specific reports
- Separated speed of system by adding a caching layer in application stack
- Provides a more structured interface than the FDA DPVs current tool
- Can consume input new report data in SQL database format.
- Is web-based to allow easy access by any employee on any type of operating system to use.
- Can organize reports into groups to be further analyzed or to be used as evidence to support a theory.

1.3 Accomplishments

The team's primary accomplishment is the development of a working prototype of a medication error and adverse reaction visualizer and organizer tool. Additionally, the team performed a local user study in order to evaluate the usability of the application and determine what future work should be done to continue the development of the tool. The application prototype has functioning front-end, back-end, cache, and database layers hosted on WPI servers. Detailed information on work finished for the application can be found below in Section 4.

Individually, each team member has learned individual coding and time management skills as well as group teamwork skills. Everyone began the project with gaps of knowledge in web development programming and has now learned how to develop with and implement all of

the technologies listed below in Section 4.1. Additionally, we all improved our teamwork skills as consequence of working as a team for six months.

2. Background and Related Work

2.1 FDA DPV Analysis Process

The layout of this interface is designed to streamline the current process for identifying, grouping, analyzing, and annotating reports. The system in place at the FDA is a spreadsheet wherein each column contains information about that specific report. While this format keeps all the reports in a structured format, there is no way to differentiate between a report that must be prioritized and a report that does not contain any relevant information. There are no visual components in the interface currently in place so implementing a treemap, timeline, and demographics section can help pinpoint trends in the reports and speed up the task of processing the important ones. Furthermore, the visualization allows investigators to filter reports into smaller and more manageable groups that can reveal patterns for specific products, stages, medication error types, etc.

The current workflow consists of four main steps. The first is to select and open a report from a list of reports. After selecting a report, the investigator examines potential signals that may be found in the report's description and summary. If any relevant information is found, the investigator then annotates the significant keywords. Once this is done, the user may share the report and start analyzing a new report.

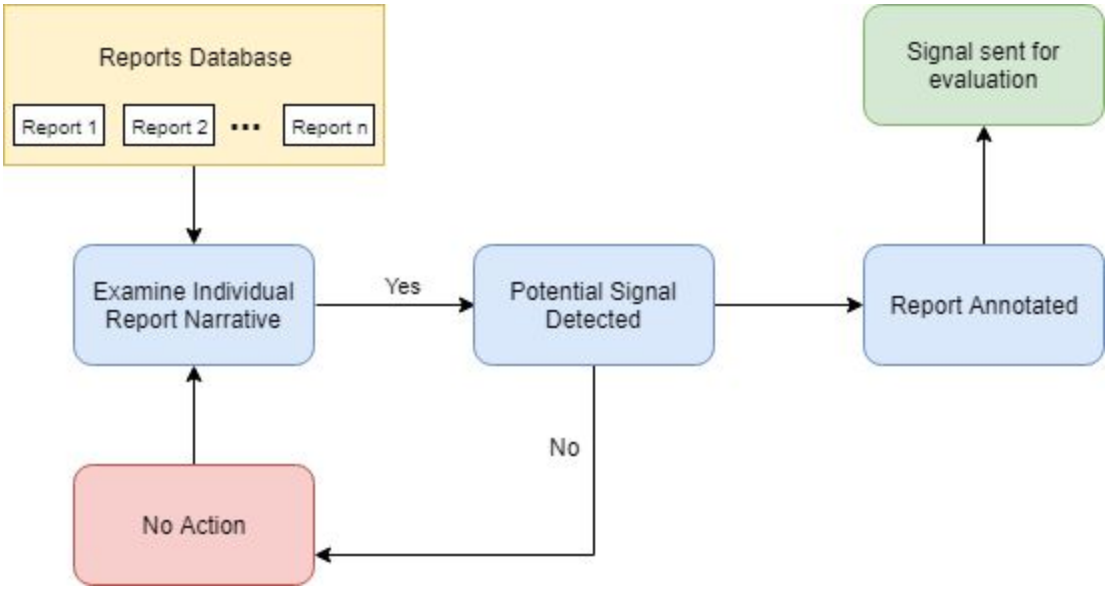


Figure 2.1 Current FDA workflow flowchart

2.2 Visuals Background

The Adverse Reaction Reports Analysis Tool compiles medication error and adverse reaction reports and visualizes them in a way that is easy to analyze and manipulate. The proposed layout by Tabassum Kakar for the interface is shown below.

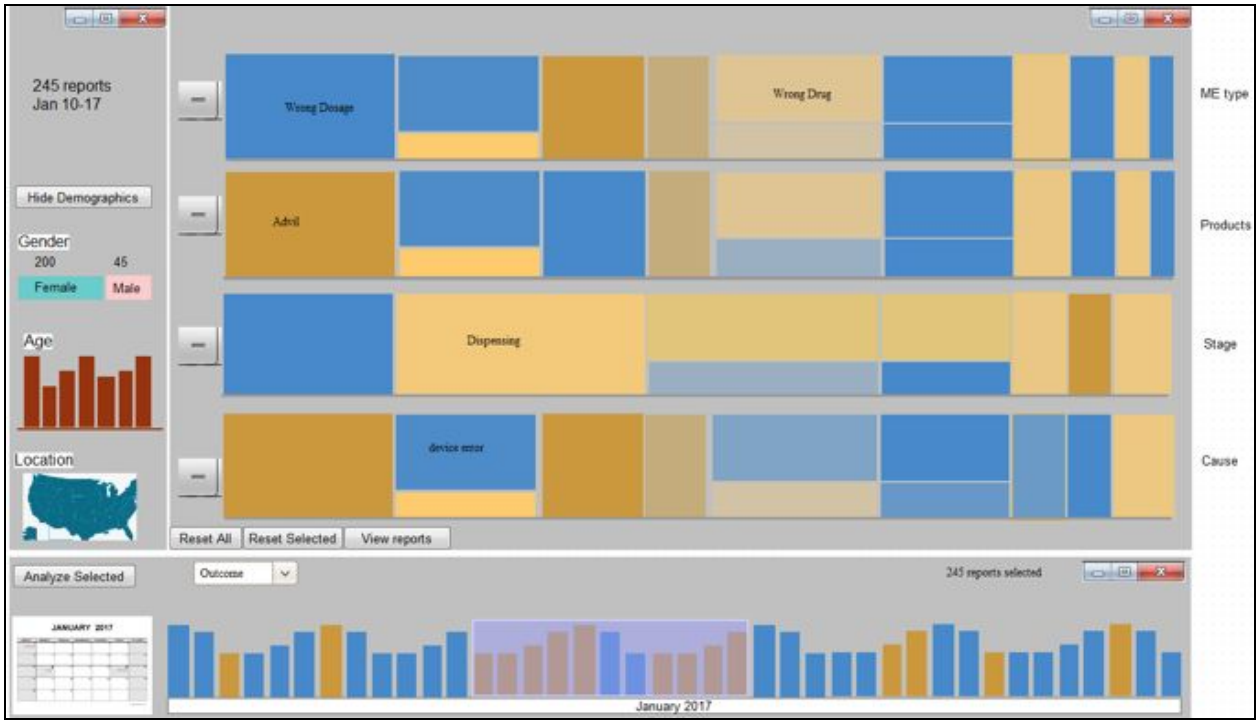


Figure 2.2: Original Proposed Layout

The layout was designed through an iterative process of repeat meetings with FDA safety evaluators about their work flow.

The layout consists of three main sections. The center portion of the interface contains tree-mappings that display information about the selected reports. A majority of the interface is dedicated to this visualization because it holds the most significant data, which investigators can use to extrapolate trends. The tree-maps display data from report medication error type, associated products, stage in which it was filed, and cause. Additionally, each tree-map contains statistics on the occurrence of these attributes that display on mouse-over. The size of each box in each tree-map is determined by the number of reports that contain that attribute. The color of each box is determined by the average severity of all the reports that contain that attribute. This allows the FDA DPV to filter the reports by report attribute as well as by report severity quickly.

On the left side of the interface is the demographics sidebar which contains visualizations for the gender, age, and location breakdowns of selected reports. The data displayed here is the summation of the demographic data associated with each report that is currently being filtered to. Each demographic graph is interactable and acts as a filter to narrow down reports further.

The lower section of the layout consists of two time-based tools- the timeline and calendar. The timeline displays how many reports were filed in a given week and changes color depending on the severity of those reports. When a time frame is selected, the visualizations in the timeline and demographics sections change accordingly to reflect the updated information. All of the sections are interactive and update dynamically to changes made in other parts of the interface.

2.3 Related Work

Several previous MQPs tackle problems similar are to those faced by the FDA DPV. The first project, titled "Context," focuses on creating a community-accessible database of an assortment of statistics and providing a tool that can visualize multiple data sets over a given timeframe. The most important component of the project was the database. After analyzing pros and cons of relational and non-relational databases, they decided to use a relational database because it offers a more structured approach and allows users to pull fewer tables when making a query [4].

Another relevant project is titled "GRID Portal Application Visualization." The goal of this project is to develop a web interface that allows users to execute parallel parametric study applications. Their application uses Java as the primary programming language and the User

Interface was made using Java Swing, a GUI toolkit. Swing was chosen over the Abstract Window Toolkit (AWT) because AWT does not allow for cross-platform compatibility. Cross-platform compatibility was essential for this project because it required that the interface allow multiple users on completely different machines to access it at all once.

The approach for this FDA project is very similar to these other projects primarily because the interface is meant to be designed for use across many platforms as a web application but we do not need to support a mass of users since this is not meant to be a publicly accessible tool.

One of the most important components of the application is the treemap visualization. Much research has been conducted to evaluate the effectiveness and accessibility of the treemap's layout. One of the benefits of the treemap is that it eliminates whitespace because every tile is part of and necessary for the visualization. Another benefit treemaps offer is that they can display entire hierarchies in relatively small spaces so users have access to a view of all levels in a hierarchy with a single visualization [9]. The layout of the treemap also facilitates the process of detecting outliers or trends in the data because the investigators' attention will naturally be drawn to the larger containers in the visualization, which typically indicate abnormalities or patterns [10].

The Adverse Reaction Reports Analysis Tool is not the first application to employ the use of treemaps, timelines, and statistics panels to convey different aspects of a specified set of data. One example of a previous interface with a similar structure is the SellTrend visual analysis tool, a system for analyzing airline travel purchase requests [11]. The SellTrend layout is largely the same as the ARRAT's layout with a couple of key differences. The interface contains a table

view where the user can see a spreadsheet-like representation of individual records. This table view is not present in the ARRAT. Instead, there is a separate page where all of the reports are listed and the user can perform several actions such as adding to a case and annotating. The UI Panel in the SellTrend interface is the equivalent of the Demographics bar in the ARRAT although the Demographics bar is positioned at the top of the application to allow for additional space for the treemap. Finally, the Timeline View in the SellTrend application is very similar to that of the ARRAT but SellTrend provides more granularity by allowing the user to view specific hours once a day has been selected. This functionality is not necessary for the ARRAT because the reports are usually marked with a date but do not provide a specific time.

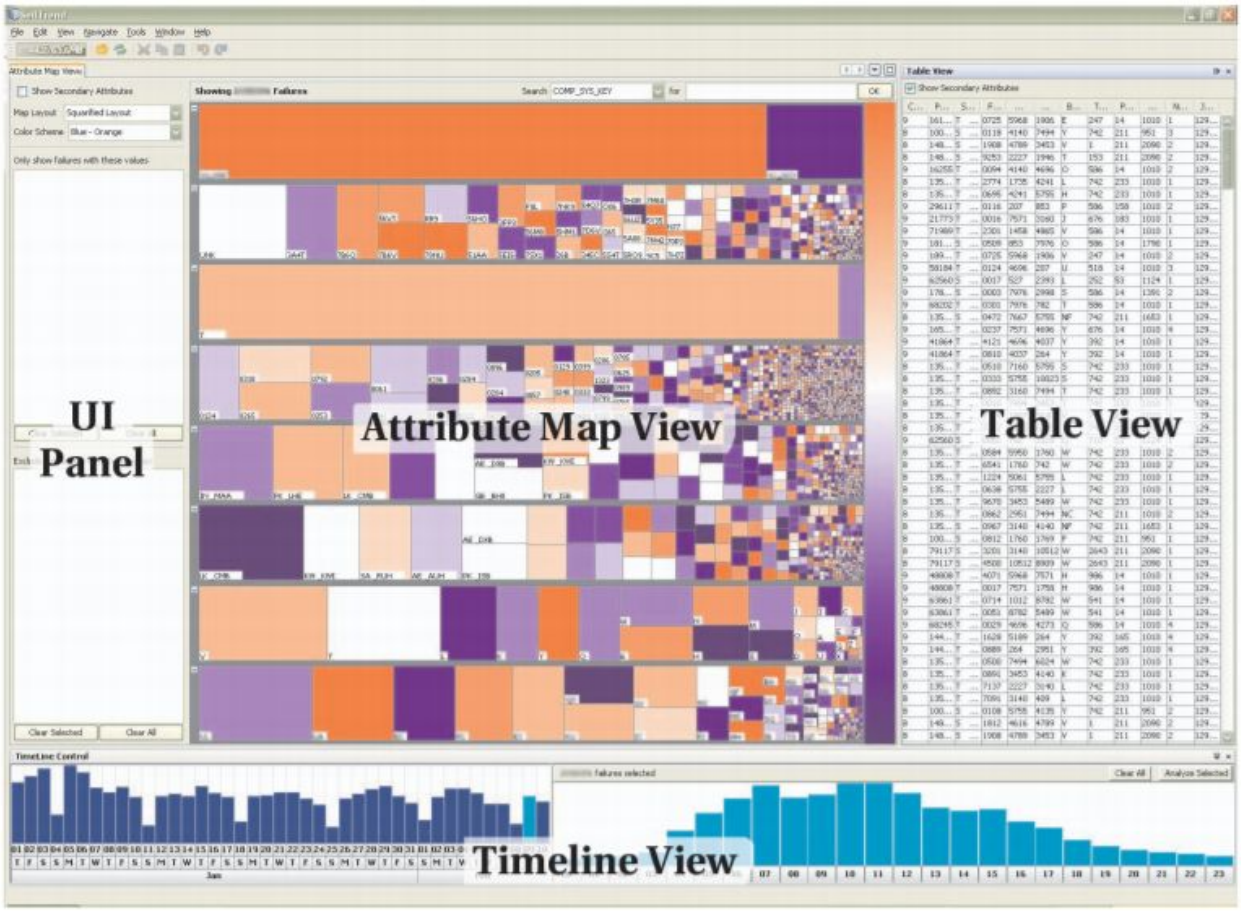


Figure 2.3: SellTrend System Interface

3. Improving Evaluator Workflow

In order to improve the workflow of FDA Drug Safety Evaluators, the team has developed a Adverse Reaction Reports Analysis Tool web application. The application allows investigators to filter and sort through reports and group relevant reports together to form cases.

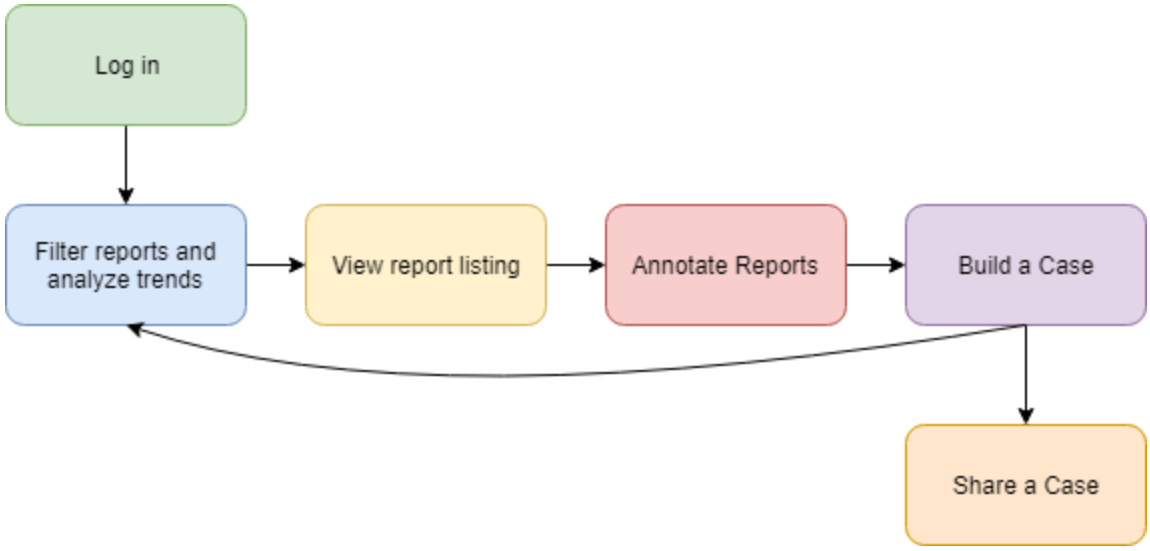


Figure 3.1 New application workflow flowchart

When an investigator first arrives at the website landing page, they are prompted to create an account as that will allow them to create and save cases of reports. Once logged in, the investigator is directed to a filtering page containing many intractable visualizations. On this filtering page, it is important that an investigator is able to filter by demographic and date as well as by the type, product or drug, stage, and the cause of a adverse reaction. This allows

investigators to narrow down their search to a handful of highly relevant reports for them to analyze.

Once an investigator has made their filter selections, they can navigate to a report listings page. Here, all of the reports that the investigator has filtered down to are shown in list form where each row is a unique report that shows that report's metadata. At the top of the page, there is a interface that lets the investigator change the contents of the report list from the filtered reports to the contents of any active case they are working on. The investigator can sort the list by any of the criteria mentioned before. While analyzing the reports, if an investigator finds an irrelevant report, they can move it into a special trash case that prevents that report from showing up again. If the investigator finds a relevant report, they can either move the report to an existing active case or create a new case for the report to me moved to. Furthermore, the investigator can open up the text of a report in order to add annotations. Annotations are not user-specific and can be viewed and changed across all investigators.

After the investigator has analyzed all the reports on the report listing page, they can go through the filtering process again or navigate to the dashboard page. The dashboard page shows all active and archived cases as well as each of their metadata and reports. On this page the investigator can view the details of each case, which includes a table of reports in that case, toggle a case to be either "active" or "archived," or edit that case. After finishing and archiving a case, the investigator can create more cases and navigate back to the filter page, restarting the whole process.

4. Application Architecture

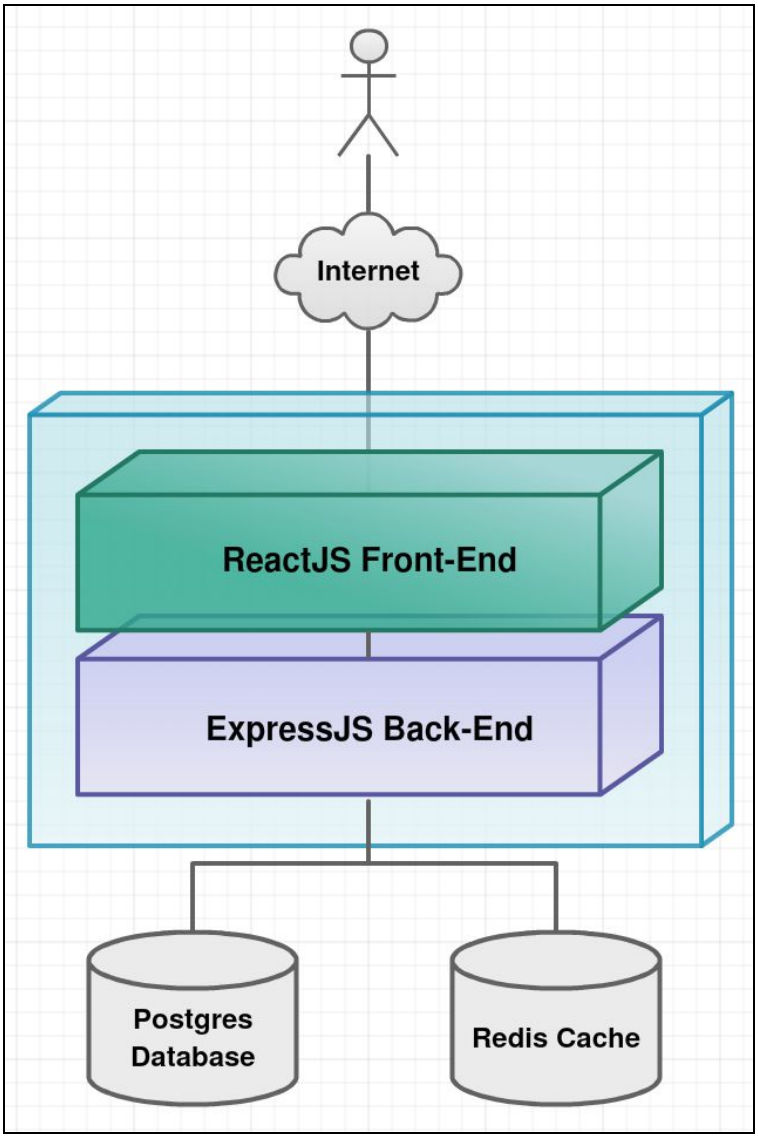


Figure 4.1: Application Stack Diagram

The Adverse Reaction Reports Analysis Tool requires three core components built before the development of visualizations: a front-end, back-end, and database client. The front-end handles the visualizations and visual effects that make the user experience immersive and

dynamic. The back-end manages data retrieval from the FAERS database and passes that information to the front-end where it is made accessible. The database stores FAERS data as well as important user-specific data used by the front-end. The front-end and back-end use the same runtime engine so they can communicate seamlessly with each other preventing hiccups during implementation.

Both the front-end and back-end use NodeJS as their runtime environments and act as event-driven servers that can send and receive GET and POST requests to communicate data back and forth. In addition to the innate ability to support cross-origin scripting, NodeJS has an extensive list of packages and libraries which allow developers to utilize a wide variety of development tools. The back-end uses Node Express - a minimalist web framework built off of NodeJS. Node Express includes the same capabilities as NodeJS and provides extra server-based functionality that is easier to implement.

The database runs Postgres because it offers many benefits for the FAERS data as the data is publicly available and formatted as relational tables. Additionally, Postgres is very efficient when dealing under two terabytes of data. The FAERS data is comprised mostly of short text values and integers and falls under that storage space threshold. Before accessing the database, the back-end queries a Redis client, an in-memory data store, that saves database query results to speed up large or repetitive queries.

The tool currently utilized by the FDA to analyze medication errors and adverse reaction reports is unintuitive and slow. The FAERS database, as mentioned in the Background, presently holds 14 million reports. This is a concern for a visualization application - considering visualization has to account for any section of that data at any time and dynamically parsing

large data sets is slow. While displaying all 14 million reports concurrently is not a common use-case, performance is still a concern. The separation of the back-end and front-end environments allow more bandwidth across each server for its intended task.

4.1 System Requirements

The project uses the following technologies [see appendix A for sources] :

- NodeJS: An open-source web server framework that runs JavaScript.
- NodeJS-ExpressJS: A web framework built on top of NodeJS that provides easy-to-use RESTful api.
- ReactJS: A JavaScript library used to build UIs.
- Redux: A state container that stores information that can be accessed by any layer of an application.
- Redux-Thunk: Redux middleware that delays the execution of a function, used to create global functions that alter the Redux state of the application.
- Redux-Persist: A Redux library that takes advantage of browser memory to persist the redux state across browser refreshes and tabs.
- React-Prop-types: A React library that enforces specific data types to be passed through specific prop variables to prevent runtime errors.
- React-router: A React library that handles URL parsing in order to deliver the correct App view to the browser.
- React-quill: A React text editor built off QuillJS- an open-source text editor built in JavaScript.

- D3: A Javascript library that processes and displays visualizations.
- D3-Recharts: A D3 module that contains powerful pre-built chart visualizations
- Lodash: A Javascript library that replaces and adds to many built in JavaScript functions with higher-efficiency versions
- Bluebird: A high-performance JavaScript promise library
- JQuery: A JavaScript html traversal and editor library
- MomentJS: A data and time parser JavaScript library.
- Webpack: A framework that bundles and parses JavaScript modules to be runnable in all browsers.
- Postgres: An open-source, relational database system.
- Redis: An in-memory key-value store that saves frequently fetched data or results of slow database queries.
- ReJSON: A Redis module that allows redis to store json as a native data type.
- Bootstrap: A front-end framework that provides easy CSS and html formatting and design
- Material-UI: A React UI library that has many built in custom elements.

4.2 Technology Choices

Throughout the development of the Adverse Reaction Reports Analysis Tool, the team had to make decisions on which frameworks and libraries to utilize to store, select, visualize and interact with the data.

The front-end of the application had to be a web-based solution. This gives the ability for multiple investigators to view and interact with the data on page that is familiar to them. With a three level structure in mind for the database, server, and web-page, the Model-View-Controller (MVC) architecture was an obvious choice. ReactJS [6] and AngularJS [7] are the two most common JavaScript frameworks with an MVC architecture implementation. While each option has their own merits, namely React having a smaller package size for faster loading, shadow DOM for reducing expensive screen re-renders, familiar syntax to HTML with JSX compiling, and Angular having TypeScript, new templating syntax, great documentation, and a more boilerplate structure etc. React was chosen as the best option. Much of this decision comes from the flexibility and familiarity of JSX when using Redux as a state manager, this leads to a faster learning curve when compared to Angular whose new 'magic' is foreign to developers at a first glance.

To keep a consistent, clean and responsive theme to the application the Material-UI Next styling library is used. This implements the Material Design Guidelines [8] created by Google that give the investigator a responsive interface with similar behavior to many of the Google apps. Using this library allows the developers to focus more on features and layout rather than implementing custom CSS for each button or menu on the page. It is built for React applications and integrates seamlessly with the architecture to serve as a tool for making a great user interface.

When developing any application it is critical to keep a clean, maintainable and understandable code-base. A tool used to set guidelines for code structure and style is ESLint.

This repository uses the AirBnB style guidelines to help all contributors write code in a standardized format, while giving helpful hints about preferred practices along the way.

4.3 Project Structure

The project is comprised of three main components the front-end, back-end, and database. The front-end has an outermost index component that houses the redux actions and reducers, as well as every other page of the application.

```
|-back-end
|-front-end
 |---public
 |____src
 | |____resources
 | |____components
 | | |____visualization
 | | | |____components
 | | | | |____demographics
 | | | | | |____components
 | | | | | | |____components
 | | | | |____timeline
 | | | | | |____components
 | | | | | | |____components
 | | | | |____treeMap
 | | | | | |____components
 | | |____portal
 | | | |____images
 | | | |____userComponents
 | | |____cases
 | | |____components
 | | |____editor
 | | | |____images
 | | |____reports
 | | | |____components
 | |____actions
 | |____reducers
```

Figure 4.2: Folder structure of the front-end

The back-end is a series of RESTful endpoints that generate and execute SQL queries then returns the json data. A REDIS cache is used to speed up the commonly used queries. These queries are processed by the PostgreSQL database containing the FAERS data.

4.4 User Dashboard

4.4.1 Front-end

In the user dashboard, an investigator will be able to see basic user and case data. The front-end development displays primarily Bootstrap-themed styling, with the help of some Material-UI navigation to allow scrollable access to their case names. On the user dashboard, investigators will be able to see who they are logged in as, how many active and inactive cases the investigator may have present in the application. They will be able to see this basic data underneath the page title ‘User Dashboard.’

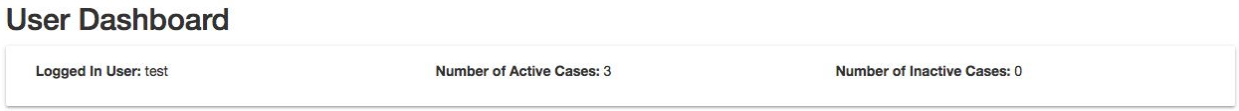


Figure 4.3: User Dashboard Basic Data

In addition to basic user and case counts, an investigator has the option to customize some of the already built cases that exist. In the next section of the dashboard, we will see the Material-UI navigation system to toggle from case to case. By clicking a tab, you will be able to see independent information about the case which is currently ‘active.’ The active tab which you most recently click on will be the case which you are seeing the details for. Details for each case include the case name, a button an option to edit the case, the case description and a non-editable listing of all the reports which belong to the investigators case.

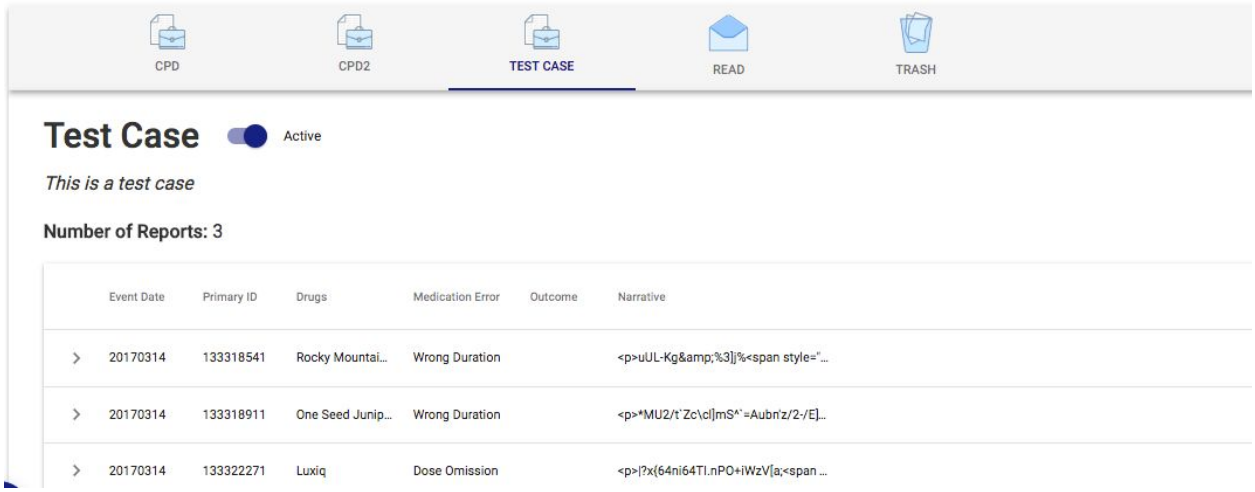


Figure 4.4: User Dashboard Case Details Listing

In addition to utilizing the Material-UI tabs component, their toggle component was used to ensure fluid animation in a react environment to toggle a case active or inactive if the case is not one of the two predefined ‘Read’ and ‘Trash’ cases.

In the report item itself, once the arrow to the left is clicked a panel will expand with the options for the reports which exist in your case, investigators will only be allowed to read the narrative for the report at hand and not move the reports from case to case nor will investigators be able to annotate the report text like they would in the Reports Listing interface.

4.4.2 Back-end

The dashboard uses a database query to get all of the case information for the currently logged in investigator that can be found in the Redux state.

```
SELECT DISTINCT name, case_id, description, active
FROM cases
WHERE user_id = 45 AND primaryid = -1
```

Figure 4.5: Get case data from the ‘cases’ table

The dashboard also makes a database query to update the status of a case when the investigator wants to set it to be active or inactive.

```
UPDATE cases
SET active = 'false '
WHERE name = 'Advil' AND user_id = 45 AND primaryid = -1
```

Figure 4.6: Get case data from the ‘cases’ table

The dashboard makes a database query to update the description or name of a particular case.

```
UPDATE cases
SET name = 'Advil', description = 'New Advil Description'
WHERE user_id = 45 AND name = 'Advil'
```

Figure 4.7: Modify a case in the ‘cases’ table

4.4.3 Database

The ‘Cases’ table is the only table used when displaying the user dashboard.

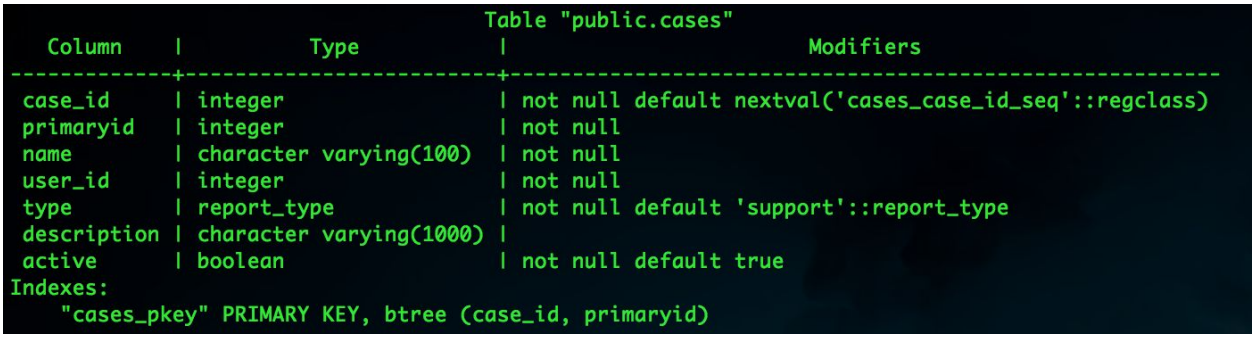


Figure 4.8: Cases Table in the database

4.5 Top Bar & Navigation

4.5.1 Front-end

The navigation bar utilizes the Bootstrap and Material-UI libraries and contains application branding, active filters, and an expandable sidebar providing navigation throughout

the application. The top bar is built with Bootstrap using its Navigation component with default styling to ensure responsiveness. The expandable sidebar is created with the Material-UI drawer component because it has a less intrusive design and intuitive control scheme.

The sidebar component is made up of Material-UI list components. These components provide default styling that adds padding around the list text so the elements to fit well visually.

Figure 4.9: Top Navigation Bar

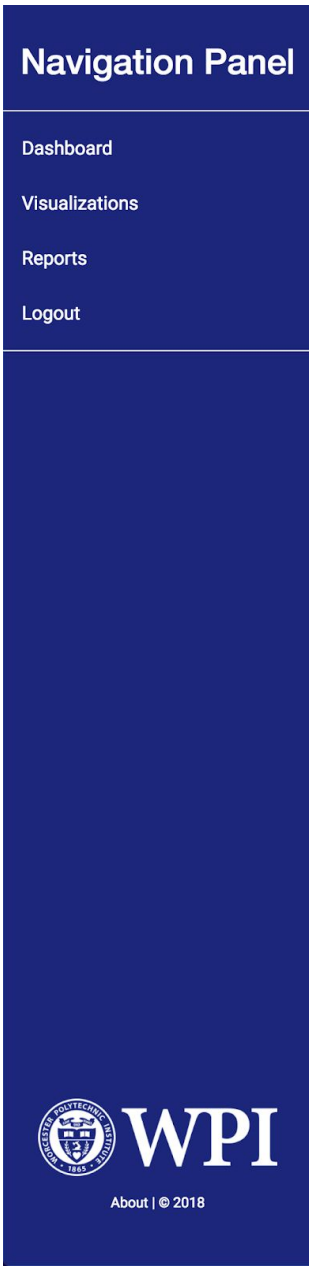


Figure 4.10: Sidebar and List Components

In addition to top navigation and sidebar navigation, the portal has three additional views- the About, Help and Login pages. The Help page contains a user manual to help investigators learn the system and can be found in Appendix F. The About page is a simple bootstrap-themed view which simply displays a page title, application title, and short project description.

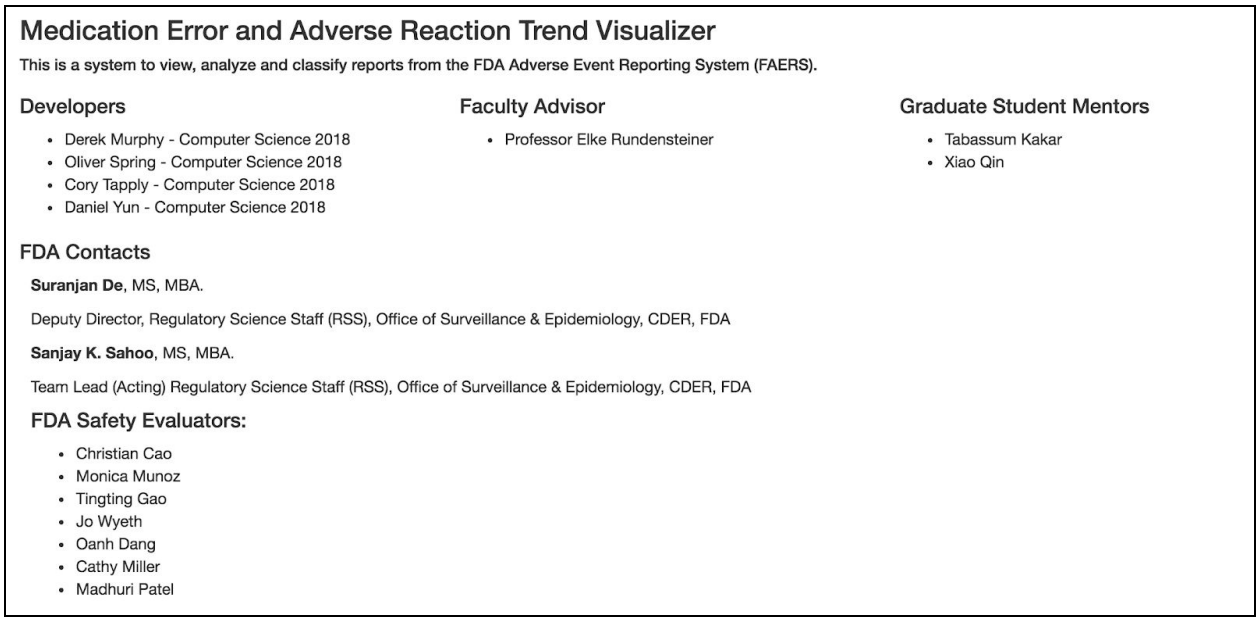


Figure 4.11: About Page

The login page contains a page title, a Bootstrap form that displays an email label, a text input field, and a "Sign In" button. User login data is sent to the Redux global state to be accessed by other views on button press.

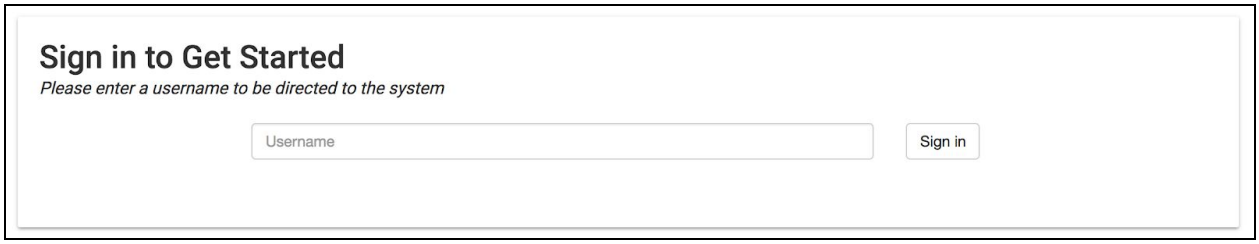


Figure 4.12: Login Page

4.5.2 Back-end

There are multiple fetch requests sent from the front-end to the back-end during a login. The login form checks the different stored emails in the 'users' table in the database and if the

user-entered email does not exist, another back-end endpoint inserts a new row into the ‘users’ table with the entered email and an auto-generated user ID number.

```
SELECT user_id, email
FROM users WHERE email=’example@example.com’
```

Figure 4.13: Get user information from ‘users’ table

```
INSERT INTO users (email) VALUES (‘example@example.com’);
```

Figure 4.14: Save user email into ‘users’ table

On user creation, the back-end generates a default ‘trash’ case that every investigator must have. The endpoint inserts a new ‘trash’ bin into the ‘bins’ table of our database that is associated with the newly generated user ID.

```
INSERT INTO bins (user_id, name, primaryid)
VALUES (1, ‘trash’ , null);
```

Figure 4.15: Save user default trash bin for report view.

4.5.3 Database

The user portal uses two columns in the ‘user’ table of the database- user_id and email. These columns identify investigators by email and tie the investigator’s email address to bins and user dependent functionalities throughout the application.

```
faers=> SELECT user_id, email FROM users WHERE email='example@example.com';
 user_id |        email
---------+---------------------
      41 | example@example.com
(1 row)
```

Figure 4.16: Users table

4.6 Main Visualization

4.6.1 Front-end

The main visualization uses a preliminary filter on the FAERS dataset based on parameters such as age, sex, date, cause, error type, etc. The main visualization is split up into three subsections: Timeline, Demographics, and TreeMap.

4.6.1.1 Timeline

The timeline is an interface that allows an investigator to easily select a time range as a data filter. It contains two methods of selecting a range.

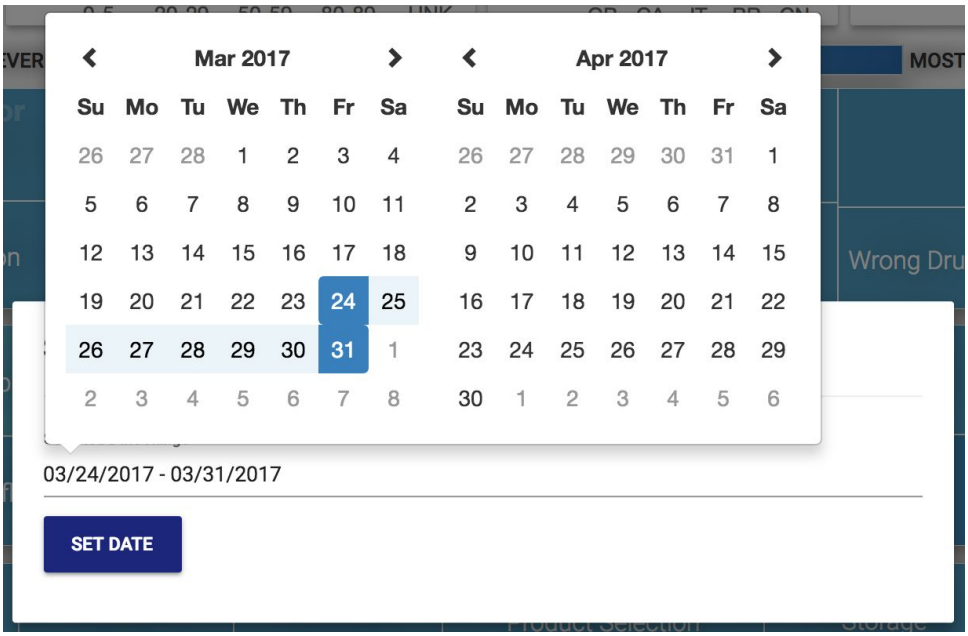


Figure 4.17: Calendar picker to select a range of dates

The calendar date-picker is created with moment.js for the underlying calendar and bootstrap + JQuery for the interface. The alternative interface is the area chart, created with

RechartsJS, that uses clicking and dragging to select the desired date range. The area chart has custom-built zoom and pan functionality to let the investigator view data effectively. The chart has two curves: the red curve represents the counts of the reports classified as serious, and the blue curve represents the reports that are not serious. The height of each curve represents the total number of reports for each date. This provides a visual representation of the amount and classification of the data right away while selecting a date range and can help discover anomalies or trends over time.

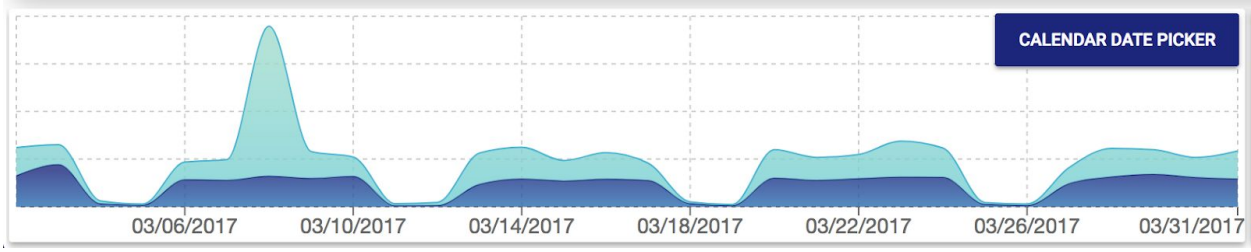


Figure 4.18: Area Chart picker to select a range of dates

Other options of charts for the Timeline are considered. For example, a bar chart is not as visually clean as an area chart.

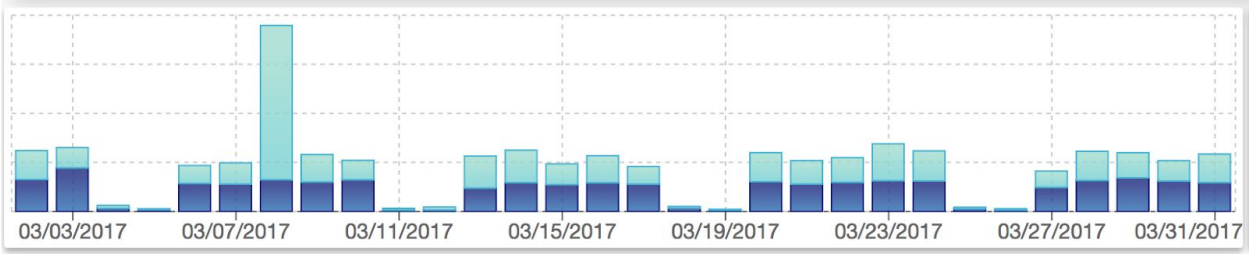


Figure 4.19: Bar Chart picker to select a range of dates

When the user hovers their mouse-over the chart they see a tooltip with exact counts of severe and nonsevere reports for that date.

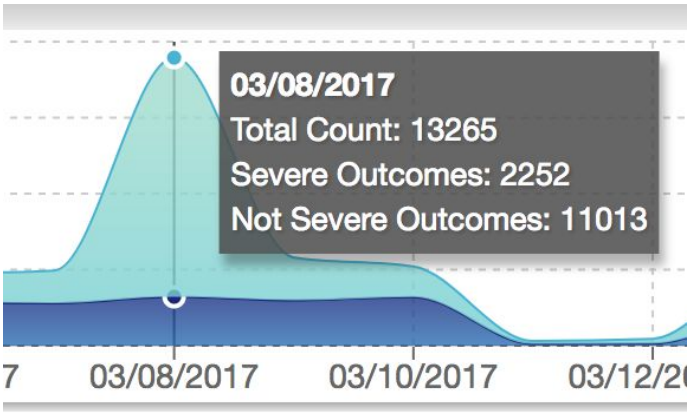


Figure 4.20: Tooltip on mouseover

4.6.1.2 Demographics

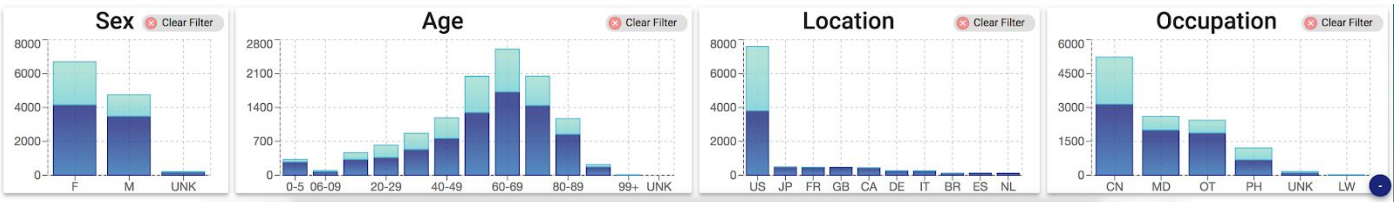


Figure 4.21: Demographics panel

The demographics panel gives the investigator the ability to see additional information about the currently selected reports. These interactive bar charts are created using RechartsJS. When a bar is clicked, the data is filtered to show reports matching that bar's criteria. As many filters as desired can be toggled on or off by clicking again on the bar itself or on the 'Clear Filter' button at the top right of each chart. Mousing over a bar displays a tooltip containing the number of reports for each different outcome code.

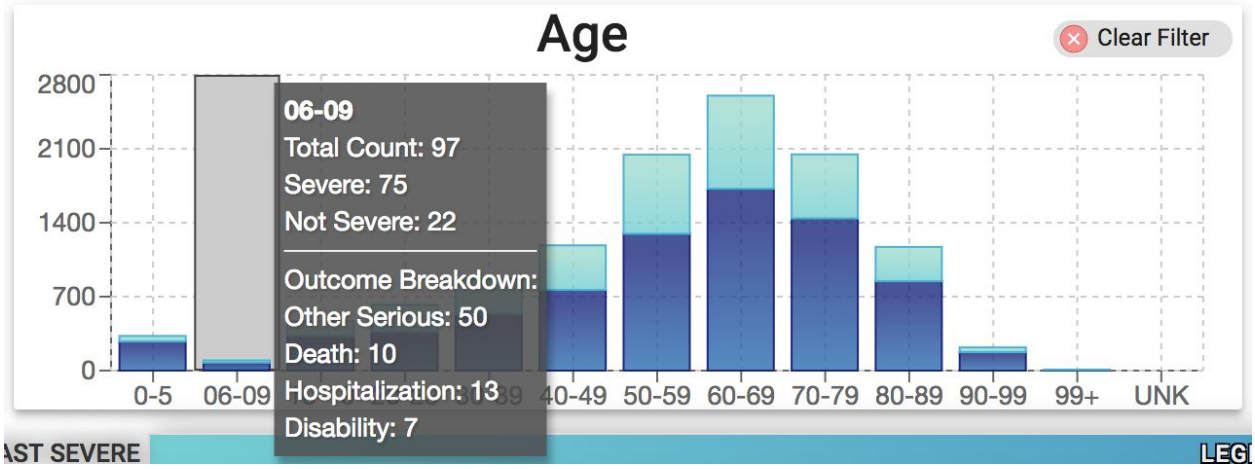


Figure 4.22: Tooltip on mouseover

4.6.1.3 Tree-Map

Figure 4.23: TreeMap

The TreeMap is built with RechartsJS and has four layers. From top to bottom, each treemap represents medication error type, product, stage, and cause. Each sub-box in each treemap can be clicked to filter out the other categories in that treemap. The color of each box represents how severe reports in that category are. Categories that are mostly severe are more red while categories with less severe outcomes are more blue. These filters can be cleared by either clicking on the box again or by clicking the ‘Clear Filter’ button on the right side of each layer.

There is a familiar tooltip displayed with each of the outcome code counts for reports in that category.

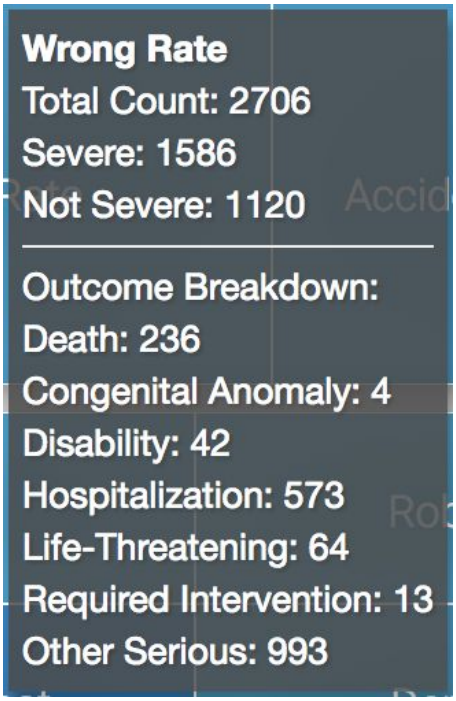


Figure 4.24: Tooltip on mouseover

To get a better view of the TreeMap, the investigator can choose to minimize the Demographics and Timeline panels to allow the TreeMap to take up more of the screen. This can be done by clicking on the circular blue minus (-) buttons on the respective panels. These buttons change to be a plus (+) to expand the panel again.

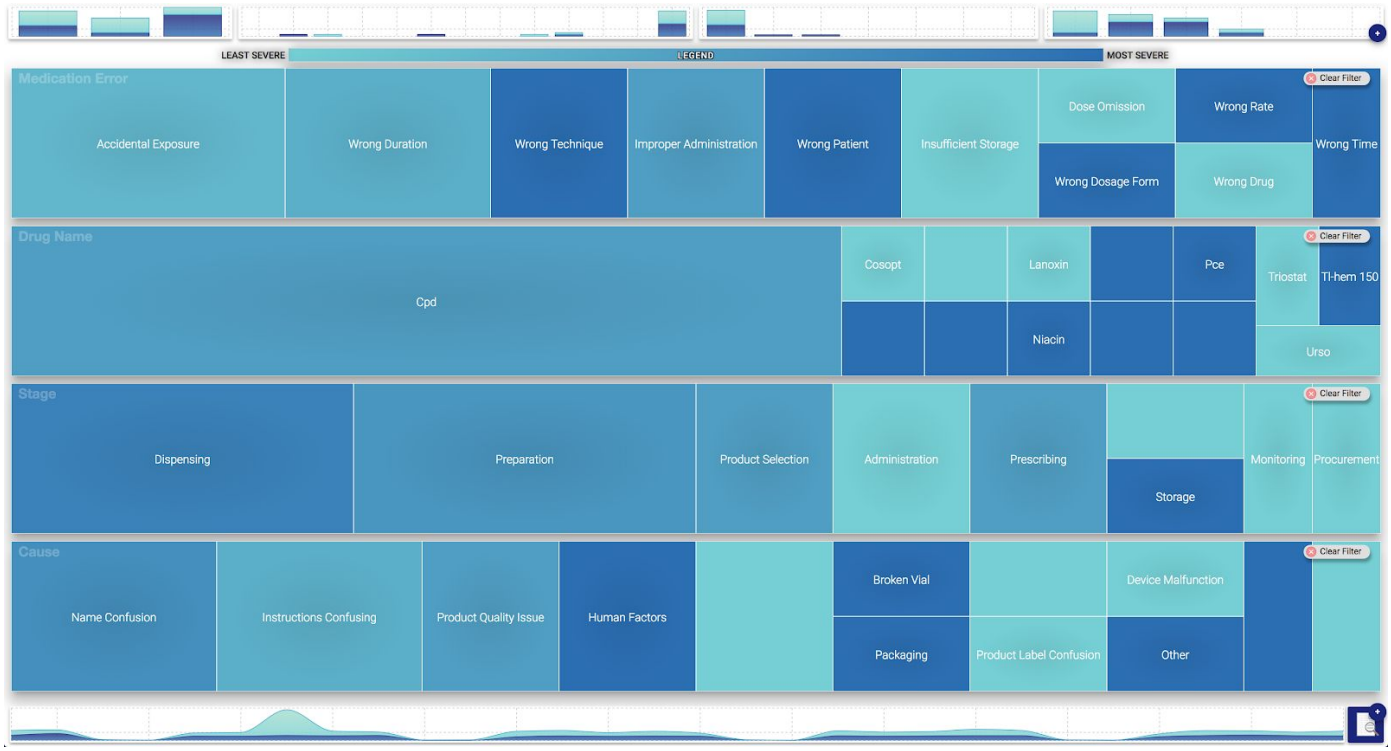


Figure 4.25: Tree-Map with Minimized Demographics and Timeline

4.6.2 Back-end

All actions made by the investigator on the interface makes a RESTful HTTP request to the NodeJS back-end. The front-end fetch request contains all of the currently selected filters to use when querying for data. The back-end generates a SQL select statement programmatically that includes all necessary filters by calling stringBuilder functions that when given a list of filter strings generates the appropriate SQL WHERE statement. These front-end requests are sent to ‘getdemographicdata’ and ‘getvis’ back-end endpoints. The ‘getdemographicdata’ endpoint returns database columns for generating the demographic charts. This data aggregates into each categories for the bar charts in the demographics panel. The ‘getvis’ endpoint handles outcome

code counts for the treemap. In this SQL statement, the database does all of the data processing and counting, increasing performance greatly. However, this process is not possible for the demographic data due to the format of the data in the database.

```
SELECT me_type as name,
 count(*)::INTEGER as size,
 count(CASE WHEN outc_cod @> '{DE}' THEN 1 end)::INTEGER as "DE",
 count(CASE WHEN outc_cod @> '{CA}' THEN 1 end)::INTEGER as "CA",
 count(CASE WHEN outc_cod @> '{DS}' THEN 1 end)::INTEGER as "DS",
 count(CASE WHEN outc_cod @> '{HO}' THEN 1 end)::INTEGER as "HO",
 count(CASE WHEN outc_cod @> '{LT}' THEN 1 end)::INTEGER as "LT",
 count(CASE WHEN outc_cod @> '{RI}' THEN 1 end)::INTEGER as "RI",
 count(CASE WHEN outc_cod @> '{OT}' THEN 1 end)::INTEGER as "OT",
 count(CASE WHEN NOT outc_cod && '{DE, CA, DS, HO, LT, RI, OT}' THEN 1
end)::INTEGER as "UNK"
 FROM reports
 WHERE init_fda_dt BETWEEN 20170313
 AND 20170317
 AND (sex = 'M' OR sex = 'F')
 AND occr_country = 'US'
 AND age_year BETWEEN 60
 AND 69
 AND (occp_cod = 'CN' OR occp_cod = 'LW' OR occp_cod = 'OT')
 AND (me_type = 'Wrong Dosage Form' OR me_type = 'Wrong Technique' OR me_type =
'Accidental Exposure')
 AND (stage = 'Other' OR stage = 'Insufficient Information' OR stage = 'Prescribing')
 GROUP BY me_type
```

Figure 4.26: Example getvis database query sent by the back-end

4.6.3 Database

All of the data in the main visualization is in the ‘demo_outcome’ table. This table is a precomputed FULL OUTER JOIN of the ‘demo’ and ‘outcome’ table to avoid calculating this join for each query improving performance dramatically. PostgreSQL indices use primarily on the ‘init_fda_dt’ column as well as the ‘sex,’ ‘age,’ ‘location’ and ‘occupation’ columns which increase performance by up to 50 times.

4.7 Reports Listing

4.7.1 Front-end

The reports listing view uses DevExtreme React Grid, a component that displays data from a local or remote source and supports paging, sorting, filtering, and grouping. The DevExtreme React Grid comes with built in Material-UI theming, which complements the rest of the application. Each of the rows represents one report in the database and the list of reports only contains those that the investigator has filtered by in the main visualization. Each row has a drop-down section where several components are housed. That section contains a button that directs the investigator to that row’s specific report text in the narrative annotation view and a button to move the report to the investigator’s trash bin. The investigator can sort by any of the table’s columns by clicking any column. An arrow appears to tell the investigator that the table is currently being sorted by the selected column. The table also contains all of the reports on the same page all at once and uses scroll to navigate through them. The top of the reports listing

view displays the bin the user is currently in and can change by button press at the bottom of the page.

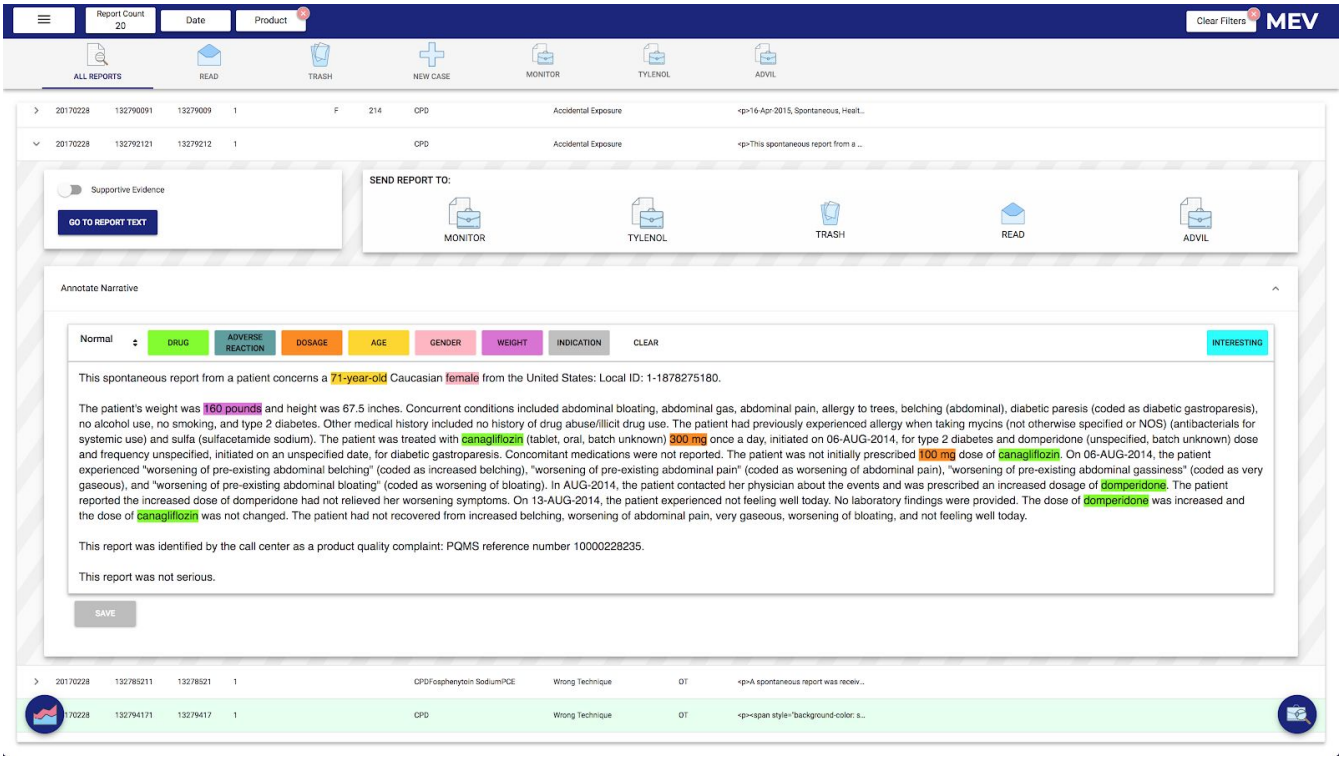


Figure 4.27: DevExtreme React Grid in the reports listing view

Prior to using the DevExtreme React Grid, React Table was used. React Table has many of the same features of the DevExtreme React Grid but one notable exception is the ability to expand rows with a drop-down menu. Sorting by columns is also built in but is much slower than the React Grid's sorting. React Table is also capable of holding all of the reports the investigator has filtered but they are divided into pages of fixed sizes so the investigator can never have all the reports in one unified table.

Name		Info		Stats
First Name	Last Name	Age	Status	Visits
television	shop	7	relationship	92
vacation	achieve	21	complicated	31
administration	war	6	single	71
quilt	smile	4	relationship	16
bottle	kick	11	single	55
gene	wish	26	relationship	45
silk	canvas	3	relationship	60
injury	medicine	20	single	68
selection	scissors	4	complicated	76
oranges	drawing	9	complicated	76

Previous Page 1 of 556 Next

10 rows

Figure 4.28: Sample React Table

4.7.2 Back-end

There are two back-end endpoints that the front-end uses. The first one is ‘getreports,’ which gets the report data the investigator has filtered so that it can be displayed in the table. The second endpoint is ‘binreport,’ which moves reports to different bins.

```
WITH bin_pids
AS (SELECT get_pid_from_bin as pids
      FROM get_pid_from_bin(user_id, ‘trash’))
SELECT *
FROM reports
WHERE (int_fda_dt BETWEEN init_fda_dt.start AND init_fda_dt.end);
```

Figure 2.29: getreports query that will return all the reports that took place between the selected start and end dates

4.7.3 Database

There are two tables in the database that the reports list view uses. The first one is the reports table. The columns in the table are all of the fields the investigator needs to see when looking at the reports list view. The wt_lb column is the weight reported converted to lbs. The second table is the bins table, which contains a user id column, a bin name column, and a primary id column. The primary key for this table is generated by combining the user id and the name of the bin. Since every investigator has a different user id and no investigator’s bin can have the same name, this ensures that no two bins will have the same primary key.

```
                Table "public.reports"
    Column     |            Type            | Modifiers
---------------+----------------------------+-----------
 primaryid     | numeric(1000,0)            |
 init_fda_dt   | numeric(8,0)               |
 caseid        | numeric(500,0)             |
 caseversion   | numeric(10,0)              |
 age_year      | integer                    |
 sex           | character varying(5)       |
 occr_country  | character varying(2)       |
 occp_cod      | character varying(300)     |
 wt_lb         | numeric(10,0)              |
 me_type       | character varying(500)     |
 stage         | character varying(500)     |
 cause         | character varying(500)     |
 drugname      | character varying[]        |
 outc_cod      | character varying[]        |
 report_text   | character varying(10000)   |
 tags          | jsonb                      |
Indexes:
    "report_idx" btree (init_fda_dt, age_year, sex, occr_country)
```

Figure 4.30: Reports Table

```
                                  Table "public.cases"
   Column    |          Type           |                        Modifiers
-------------+-------------------------+---------------------------------------------------------
 case_id     | integer                 | not null default nextval('cases_case_id_seq'::regclass)
 primaryid   | integer                 | not null
 name        | character varying(100)  | not null
 user_id     | integer                 | not null
 type        | report_type             | not null default 'support'::report_type
 description | character varying(1000) |
 active      | boolean                 | not null default true
Indexes:
    "cases_pkey" PRIMARY KEY, btree (case_id, primaryid)
```

Figure 4.31: Cases Table

4.8 Report Narrative Annotation

4.8.1 Front-end

The narrative annotation view uses React-Quill to edit report text. The library provides common text editor functionality including font size, bolding, italicizing, underlining,

strikethrough, font color, and text highlighting. Quill works with text in html format, which makes the back-end processes straightforward. The alternative is using formatting offsets, which can be very error-prone. While Quill does have built in history (undo/redo) functionality, it does not have a built in autosave feature. To implement an autosave, the front-end react state holds two versions of the text: one stores whatever is inside the text editor box in real time, and the other stores the report text that exists in the database.

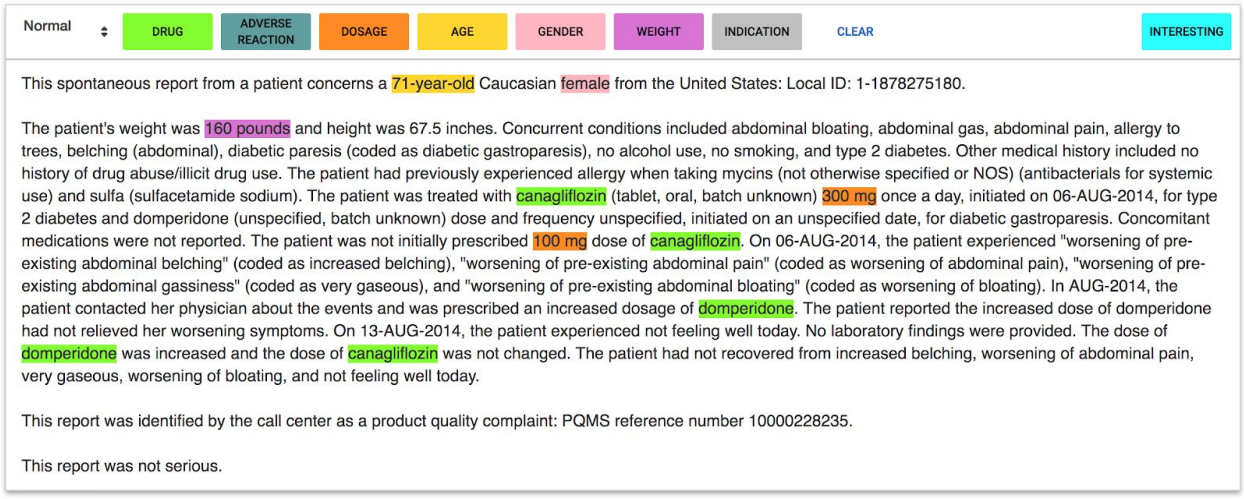


Figure 4.32: Text Editor Text

This spontaneous report from a patient concerns a 71-year-old Caucasian female from the United States: Local ID: 1-1878275180.
 The patient's weight was 160 pounds and height was 67.5 inches. Concurrent conditions included abdominal bloating, abdominal gas, abdominal pain, allergy to trees, belching (abdominal), diabetic paresis (coded as diabetic gastroparesis), no alcohol use, no smoking, and type 2 diabetes. Other medical history included no history of drug abuse/illicit drug use. The patient had previously experienced allergy when taking mycins (not otherwise specified or NOS) (antibacterials for systemic use) and sulfa (sulfacetamide sodium). The patient was treated with canagliflozin (tablet, oral, batch unknown) 300 mg once a day, initiated on 06-AUG-2014, for type 2 diabetes and domperidone (unspecified, batch unknown) dose and frequency unspecified, initiated on an unspecified date, for diabetic gastroparesis. Concomitant medications were not reported. The patient was not

initially prescribed 100 mg dose of canagliflozin. On 06-AUG-2014, the patient experienced "worsening of pre-existing abdominal belching" (coded as increased belching), "worsening of pre-existing abdominal pain" (coded as worsening of abdominal pain), "worsening of pre-existing abdominal gassiness" (coded as very gaseous), and "worsening of pre-existing abdominal bloating" (coded as worsening of bloating). In AUG-2014, the patient contacted her physician about the events and was prescribed an increased dosage of domperidone. The patient reported the increased dose of domperidone had not relieved her worsening symptoms. On 13-AUG-2014, the patient experienced not feeling well today. No laboratory findings were provided. The dose of domperidone was increased and the dose of canagliflozin was not changed. The patient had not recovered from increased belching, worsening of abdominal pain, very gaseous, worsening of bloating, and not feeling well today.
 This report was identified by the call center as a product quality complaint: PQMS reference number 10000228235.
 This report was not serious.

Figure 4.33: What Quill Stores

There is a timed loop that checks the equality of the last stored state in the database and the current state of the report text to see if current text should be sent to the back-end. The save button allows the investigator to accelerate this process. During the save process, the front-end checks the tags of the report text html for highlighting and stores highlighted words separately from the rest of the text as tags for that document in the database. The front-end pulls the report id out of the URL in order to build the fetch request to the back-end for the initial report text. If there is no id or an invalid non-number id, the front-end displays a user-friendly error.

4.7.2 Back-end

The queries for the narrative annotation are simple because each query only needs two columns of one row in one table. The back-end has two endpoints for the front-end to make fetch calls to. The ‘getreporttext’ endpoint queries the database for the report text of a specific report id and the ‘savereporttext’ endpoint updates the report text and tags of a specific report id.

```
SELECT report_text, tags
FROM demo
WHERE primaryid = 130776901
```

Figure 4.34: Retrieve Report Narrative Query for Report with id 130776901

```
UPDATE demo SET report_text = $$<p>test report text</p>$$, tags = (null)
WHERE primaryid = 130776901
```

Figure 4.35 Save Report Narrative Query for Report with id 130776901

4.8.3 Database

The narrative annotation editor uses two columns of the 'demo' table in the database- 'report_text' and 'tags.' 'Report_text' contains an html formatted string or null if empty. Tags contains a json object that holds any number of tag key-value attributes.

```
                                report_text                                 |        tags
----------------------------------------------------------------------------+--------------------
 <p>Test <span style="background-color: gold;">Narrative</span> Text</p> | {"yellowTag": "t"}
```

Figure 4.36: Database report_text and tags Columns for Report with id 130776901

4.9 Case Management

Case management is a large part of the ARRAT system. Cases are where investigators can collect evidence towards a signal to be further analyzed as a group to discover meaningful information. In the system each user account is given two default cases, trash and read.

Figure 4.37: Default trash and read cases

The trash case is used when the investigator determines a report has no meaningful information that can be used for any signals. For example, this could mean the narrative was ambiguous or is missing key information. When a report is sent to the trash, it is removed from all other cases it is in and will no longer appear in the 'All Reports' table. When the investigator comes across a report that may contain useful information, but is not relevant to the cases they are currently investigating, that report is sent to the read case. This colors the report in the Report Listing as a light grey to indicate it has already been looked at.

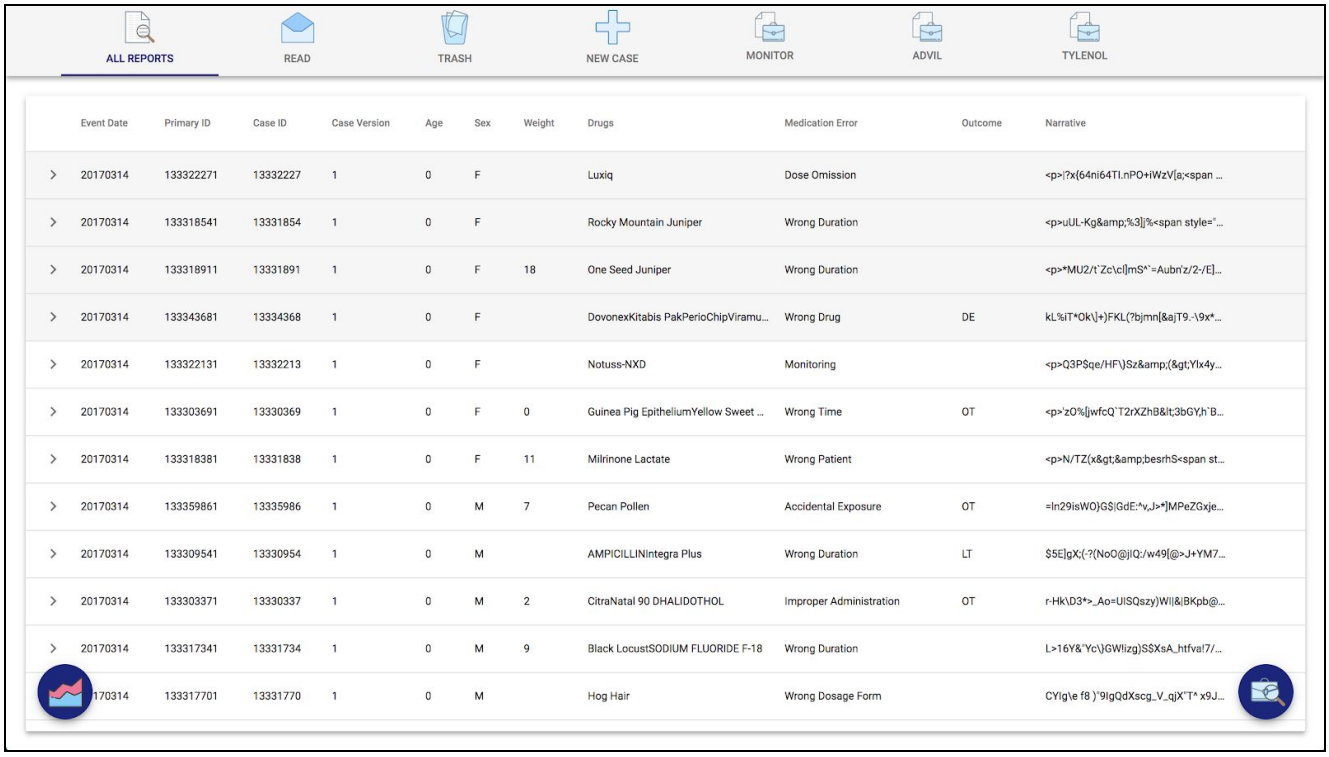


Figure 4.38: Reports Listing with reports in the read case

When a report is inside of any other case that is created by the investigator, the row in the table is highlighted green to indicate this report is inside of a case already. This allows

investigators to quickly see what reports still need to be looked into further since they are still white in the listing.

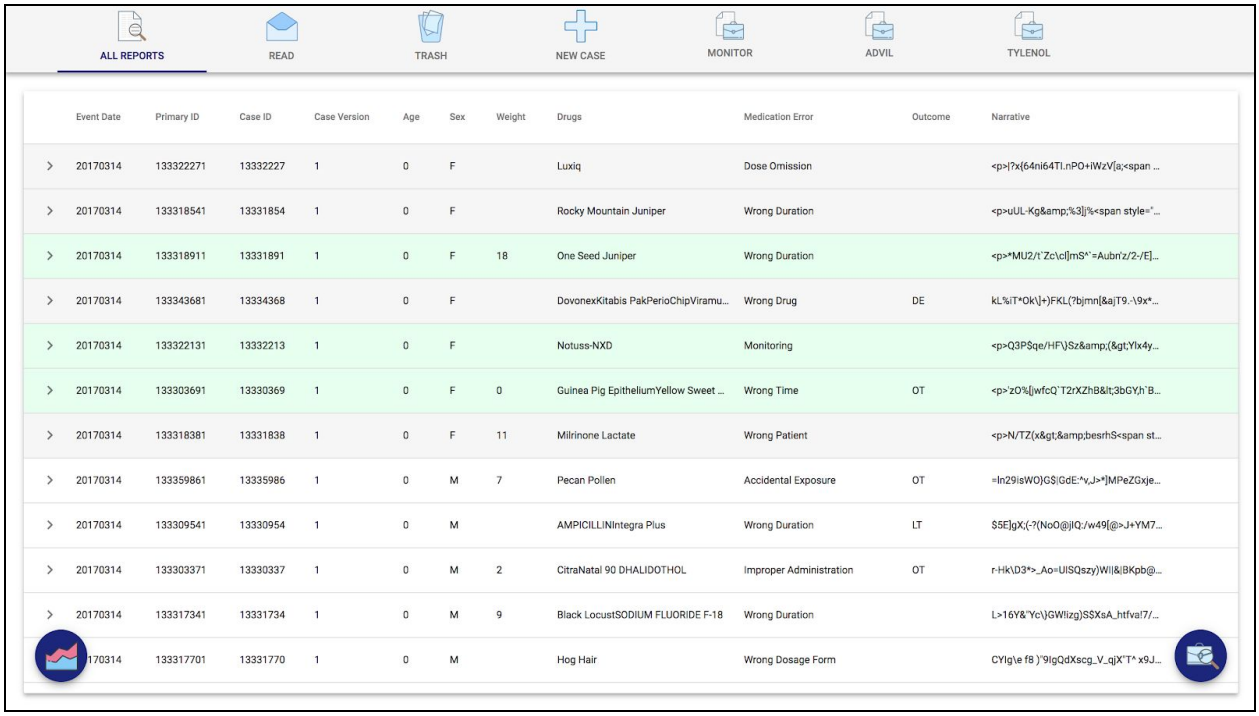


Figure 4.39: Reports Listing with reports in the read case and advil

Investigators can create a new case by clicking on the ‘NewCase’ tab at the top of the Reports Listing

Figure 4.40: New Case Tab

For example, if there is some suspected problem with some drug, Advil. An investigator can create an ‘Advil’ case and begin to collect evidence towards this suspicion. When enough evidence has been collected and the case is complete, the investigator can choose to deactivate this case in their Dashboard.

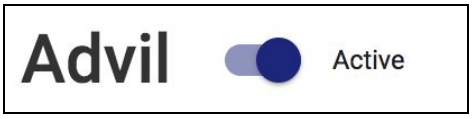


Figure 4.41: Toggle for deactivating a case

A deactivated case will no longer be shown in the Reports Listing or Case Summary Listing. There is currently no option to delete a case entirely, but an investigator can rename and change the description to repurpose a case if desired.

4.10 Case Summary

4.10.1 Front-end

The Case Summary Listing is accessed within the Reports Listing by clicking on the floating action button on the bottom right of the screen.

Figure 4.42: Floating button to open case summary

A side panel will slide in and show all of the investigator's active cases as collapsed expansion panels.

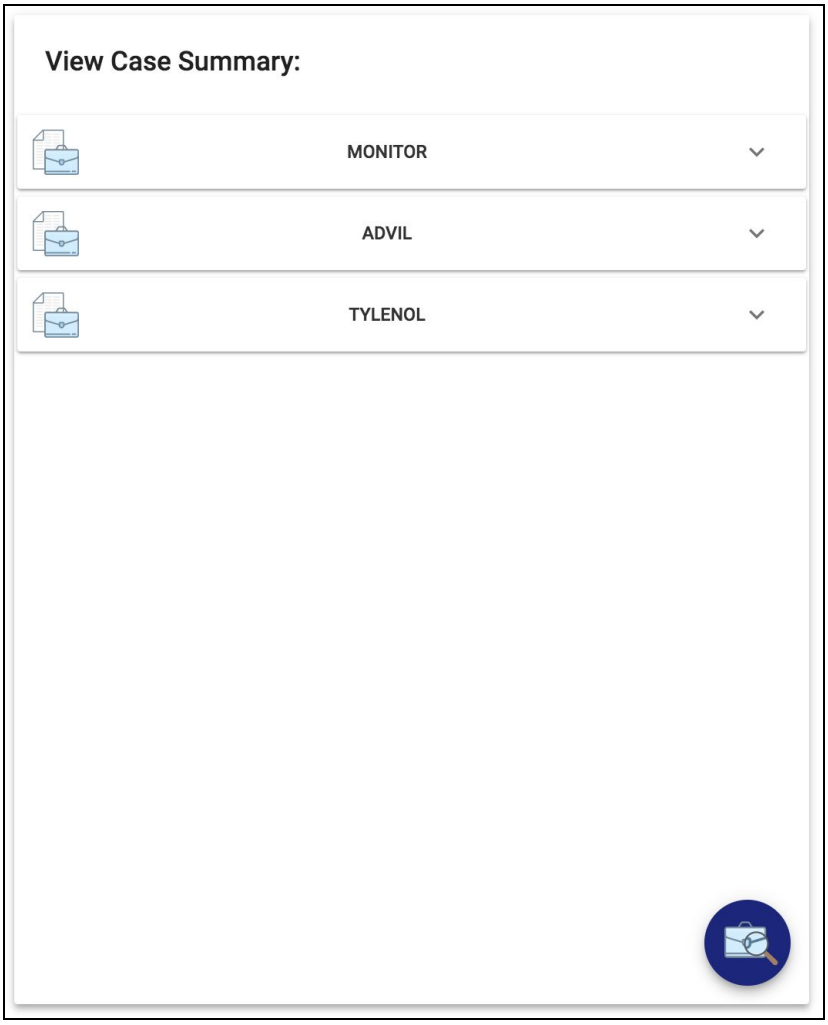


Figure 4.43: Case summary with closed expansion panels

When these expansion panel are open information such as the count of each keyword highlightings from the report narratives within the case. Also shown there is a pie chart to show how many of the reports in the case are marked as primary or supportive evidence.

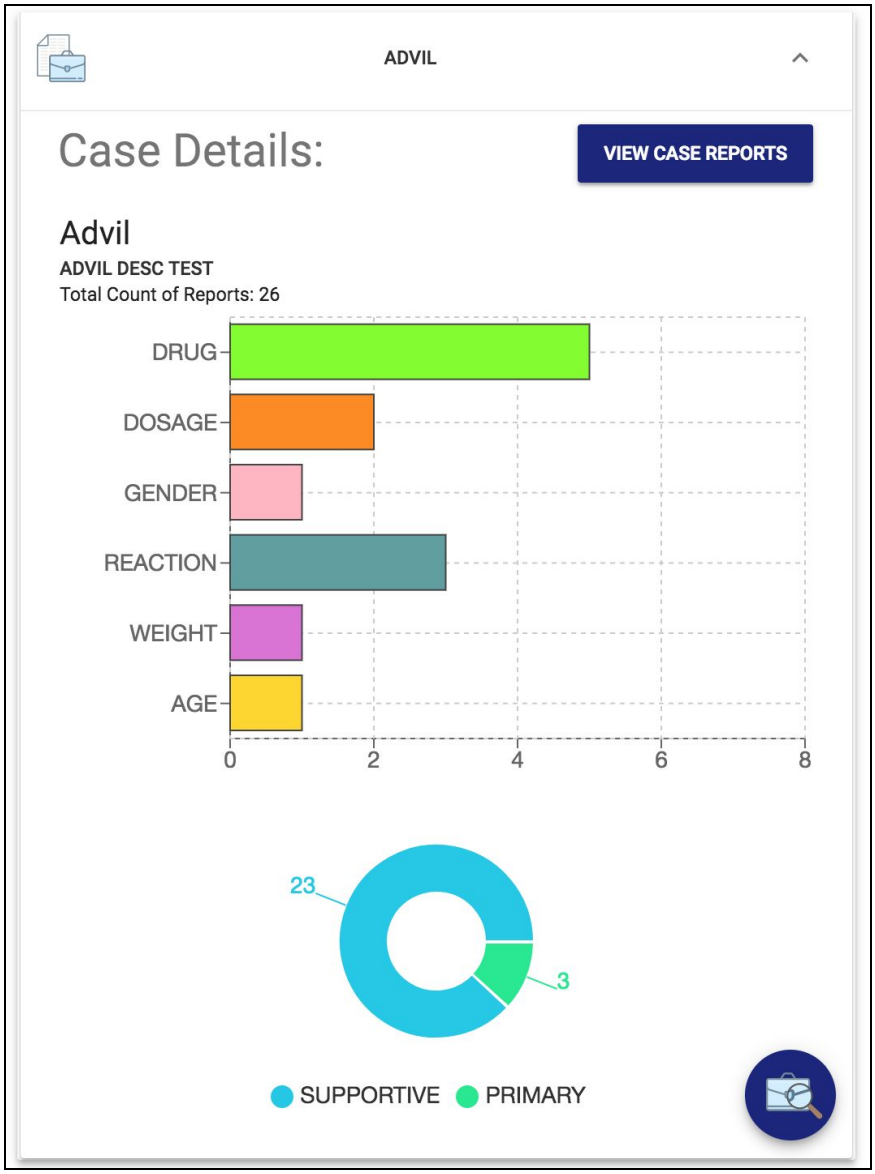


Figure 4.44: Expanded Case Summary showing data about the case

To see all of the reports inside of a particular case given the current state of the application's filters press the 'View Case Reports' button at the top of the case summary. This has the same functionality as clicking on the tab corresponding to that case on the top of the 'Reports Listing'.

4.10.2 Back-end

In order to get the data required for the Case Summary a new database query was needed to get all of the highlighted words, called tags for reports that are within each of the investigators cases.

```
SELECT tags
FROM (
   SELECT primaryid
   FROM (
      SELECT user_id, name
      FROM cases
      WHERE case_id='60'
         AND primaryid='-1' LIMIT 1
   ) as u
   INNER JOIN cases as c
   ON u.user_id = c.user_id
   AND u.name = c.name
   WHERE NOT primaryid='-1'
) as p
INNER JOIN reports as r
ON p.primaryid = r.primaryid
WHERE NOT tags IS NULL;
```

Figure 4.45: getcasetags query to get the highlighted text from all reports in a given case and user

Another query was needed to get all of the report information for the reports in the case. The '*/getreportsincases'* endpoint accepts an optional parameter to specify a single case to get reports for rather than getting reports in call cases when this parameter is left undefined.

```
SELECT DISTINCT name, *
FROM cases
WHERE user_id='351'
AND NOT primaryid='-1'
AND active=TRUE
AND name='advil';
```

Figure 4.46: getreportsincases query with optional parameter to get reports from a single case

4.10.3 Database

There was no new database infrastructure added to create this feature, all of the required database structure can be found above in section 4.8.3.

5. Evaluation Via User Studies

After finalizing development, the team began to evaluate the application through the use of user studies. The main goal of the evaluation was to compile feedback about the application's ease of use and intuitiveness.

5.1 User Testing Procedure

The user studies involved the investigators sitting down with the participants and reading the consent form (Appendix B) and asking them to sign it. After the consent form was signed, the investigators read the Adverse Reaction Reports Analysis Tool Introduction (Appendix C) to the participants to explain the purpose and functionality of the system. The next step was to give the participants five tasks (Appendix D) to complete using the ARRAT in order to evaluate the different aspects of the application. The five tasks were:

- Part 1, Familiarizing with the Filtering Methods
- Part 2, Creating and adding to a case
- Part 3, Annotating a report
- Part 4, Finding a specific report
- Part 5, Case Management

Each task has clearly-delineated steps that the participant must take in order to complete that task and the testers had questions to ask the participants to test their comprehension. After completing each task, the participants were asked to answer questions regarding their experience completing that specific task (Appendix E). The investigators recorded all of the responses in a User Study Data spreadsheet (Appendix G) and the comments they provided are listed on Appendix H.

5.1.1 Task 1- Familiarizing with the Filtering Methods

The purpose of the first task of the user study is to get participants familiarized with the different components in the main visualization page and how to interact with the data. Below is the full list of steps the participants were asked to complete.

1. Navigate to the MEV visualization page at mev.wpi.edu:3001.
2. Log into the system using your email address.
3. Using the timeline visualization, select the month of data from **1/15/2017 - 2/12/2017.**
   - What general trends do you see in this month of reports?
4. Now select the week of data from **1/29/2017 - 2/04/2017**.
   - Estimate the proportion of reports were **severe** during the week?
   - Which **drug(s)** had the most number of reports?
   - Which **medical error** (ME) has the most severe outcomes?
5. On the treemap visualization, select any **one drug** and any **two medication error types**.
6. Using the demographics visualizations, select the **gender** with the most **severe** reports.
   - What age group has the most total reports? How many?

[Ask participants questions related to Task 1]

5.1.2 Task 2 - Creating and Adding to a Case

The second task is designed to test another core function of the application: creating cases and adding reports to the cases. The participants were asked to apply specific filters and navigate to the reports listing page to create a new case and add five reports to that case. Below is the full list of steps the participants were asked to complete.

1. Navigate to visualization page.
2. Clear all the filters
3. Filter the reports down to **2/26/2017 - 3/04/2017**.

4. Filter to reports from the **United States** that contain **'CPD'** drug
    - What is the cause with the most severe reports?
    - Which age group has the most severe outcomes?
5. Navigate to the reports listing page.
6. Create a new case named **CPD**.
7. Add 5 reports to your new case and 5 reports to the trash

[Ask participants questions related to Task 2]

5.1.3 Task 3 - Annotating a Report

The third task of the user study has the participant navigate to any report's narrative annotation page to select a report and annotate it and then move it to the case they created in task 2. Once they have annotated and moved the report, they must open the case summary and view the information it contains. Below is the full list of steps the participants were asked to complete.

User Study Tasks: Part 3, Annotating a report

1. Navigate to any report's text annotation page.
2. Highlight different text in three different colors.
3. Close the text annotation page, add the report to your **CPD** case and navigate to the user dashboard.
4. View the case summary of your **CPD** case.
    - What is the most highlighted narrative keyword type in the case?

[Ask participants questions related to Task 3]

5.1.4 Task 4 - Finding a Specific Report

The purpose of the fourth task to combine the functionality of the three previous tasks to assess the users' understanding and the intuitiveness of the application. Below is the full list of steps the participants were asked to complete.

1. Navigate to the Visualizations page.
2. Clear all filters
3. Use the filters to find a report where a **75** year old **female** patient has reported an error with **Perrigo Hydroquinone** during **Administration** stage with a cause of **Product Quality Issue** on **12/27/2016**.
4. View this exact report inside of the reports listing page.
5. Highlight all mentions of a drug in the narrative.

[Ask participants questions related to Task 4]

5.1.5 Task 5 - Case Management

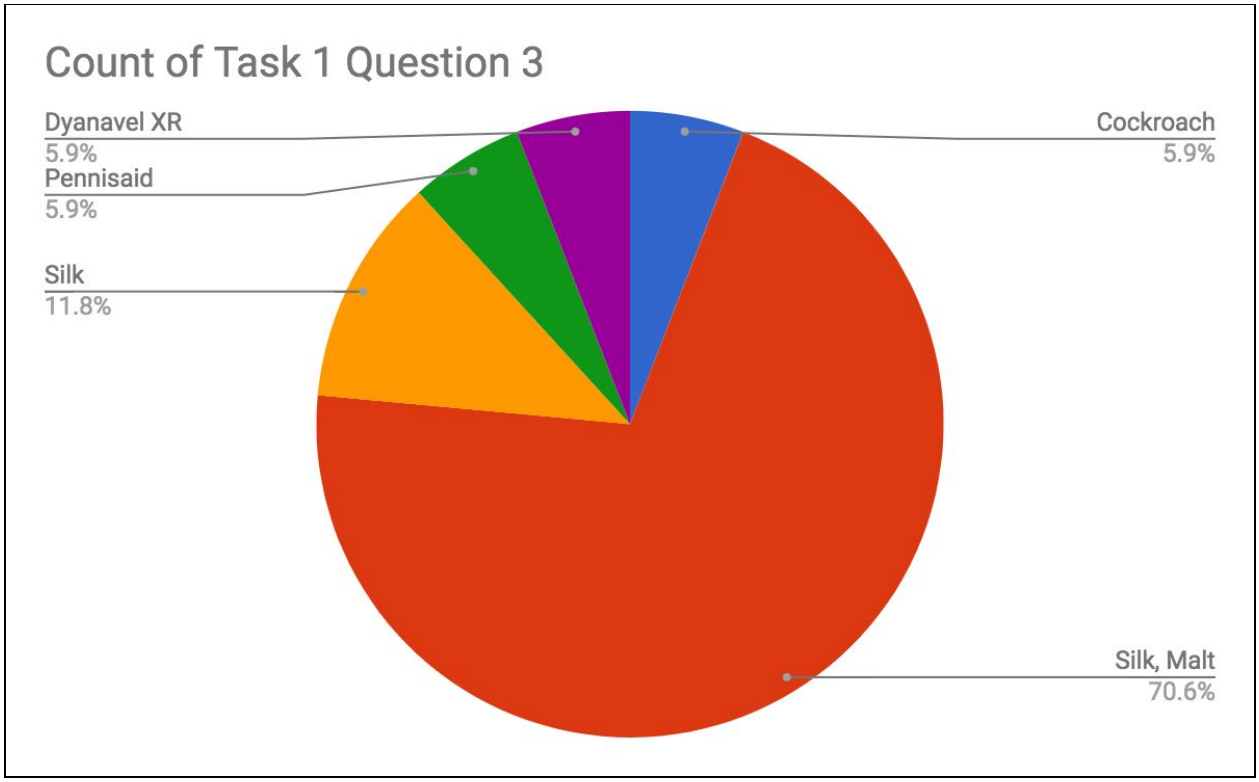
The fifth and final task has the participants log in with an account that has predetermined cases, reports, and annotations to test how easily accessible and easy to understand the data about each specific case is. Below is the full list of steps the participants were asked to complete.

1. Log in with the email 'mev.casestudy@wpi.edu'
2. Find the case with highest number of reports as supporting evidence?
   - What are 3 annotations from analysts on reports under the case **Advil**?

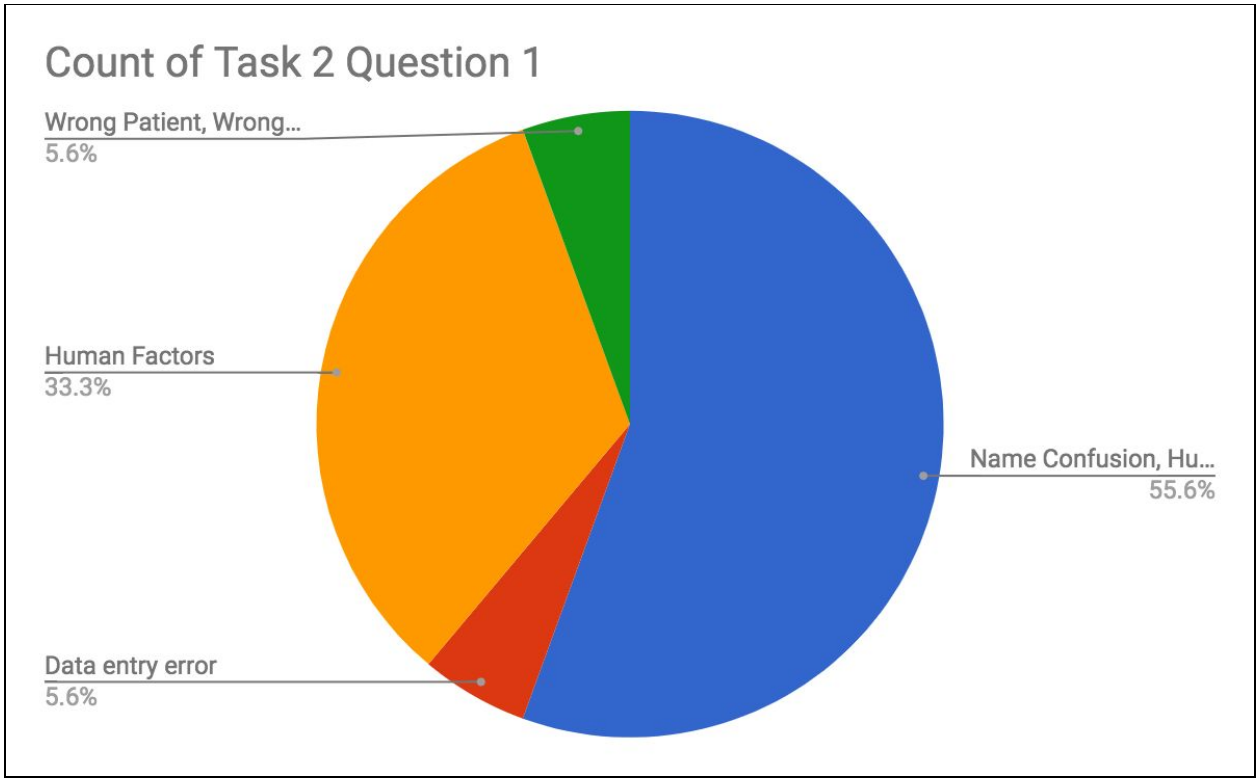
[Ask participants questions related to Task 5]

5.2 Evaluation Results & Analysis

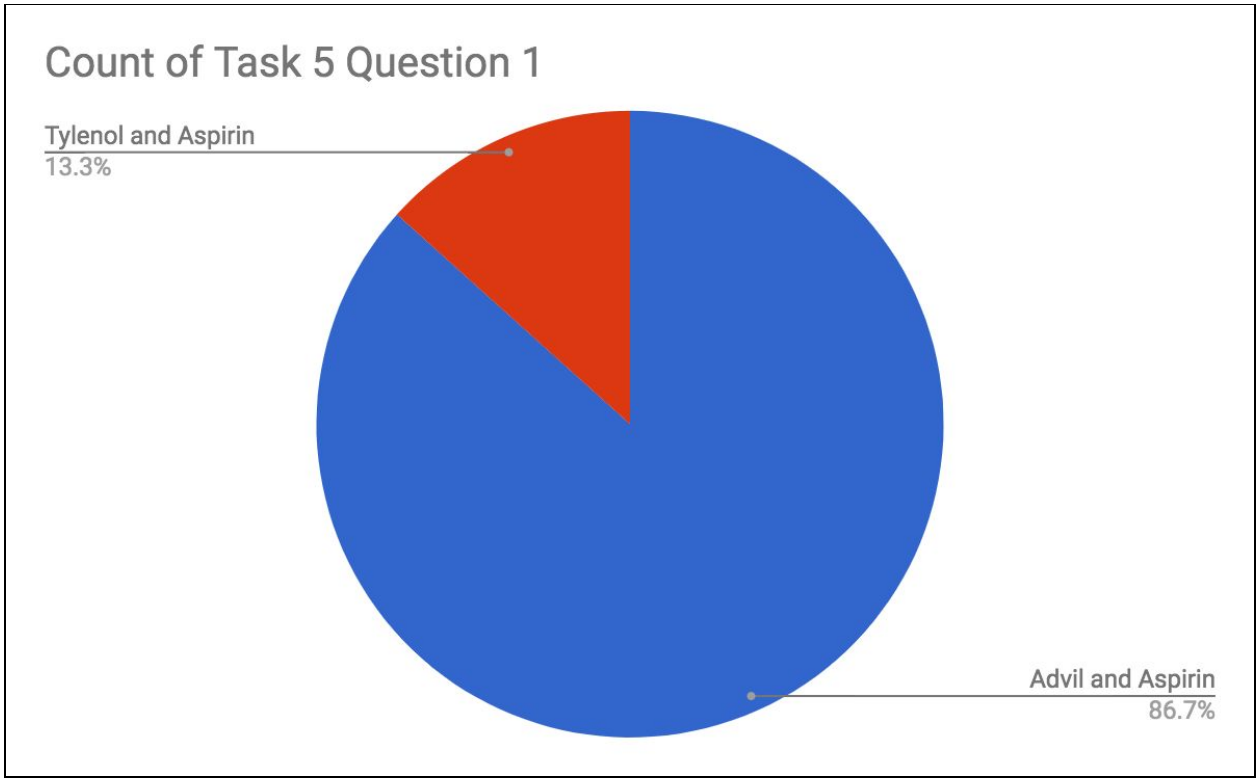
A total of 18 participants took part in the user testing process. The participants all came from diverse backgrounds and had varying experience with computers, technology, and web interfaces. The results of their answers to the tasks and post-task questions were recorded and compiled. Below are the findings.



The correct answer for question 3 of task 1 was “Silk and Malt”. The results show that 70.6% got the correct answer and an additional 11.8% got the answer partially correct. 17.7% of participants gave incorrect answers.



The correct answer for question 1 of task 2 was “Name Confusion and Human Factors”. After following the study’s steps, 55.6% of participants arrived at the correct answer and an additional 33.3% gave a partially correct answer. Only 11.2% of participants did not give one of the expected answers.



For the first question of task 5, 86.7% of participants gave the correct answer, which was "Advil and Aspirin". The remaining answers were "Tylenol and Aspirin" and are partially correct because the participants listed Aspirin.

Most frequent comments about the application:

- Timeline is unintuitive and hard to use
- Trouble setting the selected date
- The report annotation process was not intuitive

The most common criticisms regarding the application's interface were about the timeline. Specifically, many participants experienced trouble scrolling through the timeline and zooming in and out to select a specific date. Other participants noted that clicking and dragging to select a date range was not intuitive and that clicking once to select a start date and clicking a

second time to select an end date would have made more sense. Many of the participants also seemed to have trouble knowing that they needed to click the SET DATE button to apply the time filter. When it came to annotating reports, several testers reported confusion when trying to create or remove an annotation. One participant mentioned he would want the ability to "select a highlight color then highlight instead of selecting then pressing the button". To remove annotations, a participant must select text that has already been highlighted and then press the CLEAR button. Some participants did not think that was intuitive and suggested renaming it to "Erase".

As part of the user study, participants were asked to rate various facets of the interface. The ratings were done on a scale ranging from 1 to 5, with 1 meaning not very intuitive/challenging and 5 indicating the task was very intuitive/challenging. The data supports many of the comments the participants had. For instance, questions 1 and 2 of task 2 ask about how intuitive it was to create a new case and to move reports between cases. The averages of those two categories were some of the highest at 3.667 and 3.389, indicating that participants for the most part found it easy to interact with cases. Similarly, there were few comments mentioning trouble using the treemap visualization and question 3 of task 1 supports this as it has a 3.5 out of 5 rating in terms of intuitiveness. Finally, it is evident that the purpose of the Adverse Reaction Reports Analysis Tool is clear even to first-time users as question 1 of task 1 had the single highest intuitiveness rating of the entire study, at 3.778.

Task, Question	Average Rating
Task 1, Question 1	3.778
Task 1, Question 2	2.556
Task 1, Question 3	3.500
Task 1, Question 4	2.778
Task 2, Question 1	2.889
Task 2, Question 2	3.667
Task 2, Question 3	3.389
Task 3, Question 1	3.333
Task 3, Question 2	2.778
Task 4, Question 2	3.059
Task 5, Question 1	2.625

Figure 5.1 Results of post-task questions

6. Conclusion

6.1 Summary of Project and Contributions

This project ended with a working prototype of a medication error and adverse reaction trend visualizer tool that will be continued to be worked on in the future. In its final state, the system contained the entire workflow flowchart outlined in Section 3. The application currently allows an investigator to:

- Filter reports from the FAERS database based on multiple different factors.

- Display and sort the filtered reports in a list view.
- View and highlight report narratives.
- Create or modify cases to organize reports into meaningful groups.

While the functionality for all these features exists in the application, the user study evaluation points to necessary future development done to make the some of the features more user-friendly.

The project was developed using agile development with one to two week development cycles. The team would meet with graduate students Xiao Qin and Tabassum Kakar as well as Professor Rundensteiner to go over development progress each week and propose the work that would be completed for the following week. Because of the nature of agile development, most features were created and then iterated on with multiple changes and major code refactors making it hard to give singular credit for each part of the system. Keeping in mind that each team member contributed to each part of the system, Cory is most responsible for the main visualization page, Daniel is most responsible for the reports listing page, Derek is most responsible for the user dashboard and portal view, and Oliver is most responsible for the backend and narrative view.

6.2 Future Work

In the future there can be many helpful features added to the system. A key feature to be added is the ability for the system to recommend reports similar to the ones within a case to help an investigator build out a case. This could be paired with machine learning to help better classify a report as supportive or primary evidence towards a case. If an investigator could search

through narratives by typing a keyword to easily find reports that pertain to their cases could speed up workflow significantly.

Another feature is in the case summary, a case could have a series of bar charts to show the frequency of specific highlighted words within the case report narratives to show the most frequent keywords at a glance.

In the reports listing, a nice shortcut would be to allow a single click on the cell in the table that contains the short preview of the narrative to pull up the full annotator for that reports narrative. Along with this, the table on the left and the case summary views should be more linked together. For example, when viewing the summary of a case, it is likely that an investigator would be interested in viewing the reports within that case as well. If the system would pull up these reports automatically when viewing the summaries, that would reduce the number of clicks required by the investigators, thus enhancing the user experience.

A setback with the public dataset used in this system is that it lacked specific location data of the reports. If this information could be more detailed this would allow the use of an interactive heatmap of the United States to see which state reports are coming from to help narrow down the filtering process by region in a more effective manner.

A feature that would help investigators better distinguish or understand the visualizations page would be to offer customization on the colors of the system. Specifically if the investigator could choose a color theme that they preferred most would be better than having a single theme for all investigators. This project was proposed at first with the idea of having multiple forms of visualizations with the TreeMap being just one option. This was put off to design and implement the investigator workflow of viewing reports, annotating narratives and building cases. In the

future, having multiple different ways to filter down and understand these reports in the main visualization could help find new interesting connections and insights on adverse reactions with medication.

References

1. Pharmacovigilance, Oxford Dictionary. https://en.oxforddictionaries.com/definition/pharmacovigilance, 2018.
2. FAERS Public Dashboard. https://fis.fda.gov/sense/app/777e9f4d-0cf8-448e-8068-f564c31baa25/sheet/7a47a261-d58b-4203-a8aa-6d3021737452/state/analysis, 2018.
3. Durasal-Durezol Mix-Up. www.ismp.org/newsletters/acutecare/showarticle.aspx?id=5, 2011.
4. Context MQP. https://web.wpi.edu/Pubs/E-project/Available/E-project-052110-154850/unrestricted/MQP_Paper.pdf, 2010.
5. GRID Portal Application Visualization MQP. https://web.wpi.edu/Pubs/E-project/Available/E-project-042705-074945/unrestricted/CS-GXS-0502.pdf, 2005.
6. React Homepage. https://facebook.github.io/react/, 2018.
7. Angular IO Homepage. https://angular.io/, 2018.
8. Google Material UI Guidelines. https://material.io/guidelines/, 2018.
9. TreeViz: treemap visualization of hierarchically structured information. https://dl.acm.org/citation.cfm?id=142833, 1992.
10. Nv: Nessus vulnerability visualization for the web. http://www.cs.columbia.edu/~riley/pdfs/vizsec2012nv.pdf, 2012.

11. SellTrend: inter-attribute visual analysis of temporal transaction data. https://www.ncbi.nlm.nih.gov/pubmed/19834168, 2009.

12. NodeJS: https://nodejs.org/en/, 2017.

13. NodeJS-ExpressJS: https://expressjs.com/, 2017.

14. ReactJS: https://reactjs.org/, 2017.

15. Redux: https://redux.js.org/, 2017.

16. Redux-Thunk: https://www.npmjs.com/package/redux-thunk, 2017.

17. Redux-Persist: https://www.npmjs.com/package/redux-persist-store, 2017.

18. React-Prop-types: https://www.npmjs.com/package/prop-types, 2017.

19. React-router: https://www.npmjs.com/package/react-router, 2017.

20. React-quill: https://www.npmjs.com/package/react-quill, 2017.

21. D3: https://d3js.org/, 2017.

22. D3-Recharts: http://recharts.org/#/en-US/, 2017.

23. Lodash: https://lodash.com/, 2017.

24. Bluebird: http://bluebirdjs.com/docs/getting-started.html, 2017.

25. JQuery: https://jquery.com/, 2017.

26. MomentJS: https://momentjs.com/, 2017.

27. Webpack: https://webpack.js.org/, 2017.

28. Postgres: https://www.postgresql.org/, 2017.

Appendix

Appendix A: Technologies Used

- NodeJS (8.4.0)
- NodeJS-ExpressJS (4.15.4)
- ReactJS (16.2.0)
- Redux (5.0.6)
- Redux-Thunk (2.2.0)
- Redux-Persist (5.3.5)
- React-Prop-types (15.5.10)
- React-router (4.2.0)
- React-quill (1.2.0)
- D3 (4.12.0)
- D3-Recharts (1.0.0-beta.7)
- Lodash (4.17.4)
- Bluebird (3.5.0)
- JQuery (3.2.1)
- MomentJS (2.19.1)
- Webpack (3.7.1)
- Postgres (9.6)

- Redis (4.0.2)
- ReJSON (1.0.1)
- Bootstrap (v3)
- Material-UI (1.0.0-beta.30)

Appendix B: User Study Consent Agreement

Informed Consent Agreement for Participation in a Research Study

Investigators: Derek Murphy, Oliver Spring, Cory Tapply, Daniel Yun, Xiao Qin, Tabassum Kakar
Contact Information: Email at mevmqp@wpi.edu
Title of Research Study: MEV: Visualizing Medication Errors in Pharmacovigilance

Introduction: You are invited to participate in a research study to evaluate the usability and functionality of a medication error visualization application. Before you agree, however, you must be fully informed about the purpose of the study, the procedures to be followed, and any benefits, risks or discomfort that you may experience as a result of your participation. This form presents information about the study so that you may make a fully informed decision regarding your participation.

Purpose of the study: With the rapid expansion of data becoming more publicly available, the importance of ranking large content has never been more relevant. Many existing rankings target general audience, yet cannot take into account individual preferences. We have developed a web application to help users customize and interpret rankings to meet their own end goals. The purpose of this study is to evaluate the usability and functionality of our application and to get feedback for improvement.

Procedures to be followed: Your participation will consist of a one-time session where you will be asked to perform four tasks using our ranking application. After performing all tasks, you will be asked to answer some questions regarding the experience of using the Medication Error Visualizer.. There are no right or wrong answers.

Risks to study participants: This study involves the following risks: minimal fatigue and eye strain from looking at a screen. There may be other risks that we cannot predict. In case of discomfort, you may choose to discontinue your participation at any time without penalty.

Benefits to research participants and others: There are no direct benefits for you to participate in this study, but you will be contributing to the creation of an intuitive and effective medication error and trend visualizer application.

Record keeping and confidentiality: The questionnaire is completely anonymous and individual answers will not be published. End results will be aggregated before publication. Records of your participation in this study will be held confidential so far as permitted by law. However, the study investigators, the sponsor or it's designee and, under certain circumstances, the Worcester Polytechnic Institute Institutional Review Board (WPI IRB) will be able to inspect and have access to confidential data that identify you by name. Any publication or presentation of the data will not identify you.

Compensation or treatment in the event of injury: No compensation will be made for injuries resulting from participation in this study. You do not give up any of your legal rights by signing this statement.

For more information about this research or about the rights of research participants, or in case of research-related injury, contact: You can contact the team in writing an email at mevmqp@wpi.edu or the Principal Investigator, Elke Rundensteiner, at rundenst@wpi.edu. In addition, you may also contact the chair of the WPI Institutional Review Board (Prof. Kent Rissmiller, Tel. 508-831-5019, Email: kjr@wpi.edu) or WPI's University Compliance Officer (Jon Bartelson, Tel. 508-831-5725, Email:jonb@wpi.edu).

Your participation in this research is voluntary. Your refusal to participate will not result in any penalty to you or any loss of benefits to which you may otherwise be entitled. You may decide to stop participating in the research at any time without penalty or loss of other benefits. The project investigators retain the right to cancel or postpone the experimental procedures at any time they see fit.

By signing below, you acknowledge that you have been informed about and consent to be a participant in the study described above. Make sure that your questions are answered to your satisfaction before signing. You are entitled to retain a copy of this consent agreement.

_____ Date: _____

Study Participant Signature

_____

Study Participant Name (Please print)

_____ Date: _____

Signature of Person who explained this study

Appendix C: Application Guide

Adverse Reaction Reports Analysis Tool Introduction:

The Main Visualization Page

Here we can see three main sections. At the bottom we have the Timeline, this is where we first start the visualization. After we select a date range by clicking and dragging on the chart we can see the rest of the graphs show the data.

We have the Demographics panel at the top for selecting information about the patient.

In the middle we have four separate TreeMap visualizations for Medication Error Type, Drug Name, Stage, Cause.

All of the graphs on this page are interactive, if you click on any of the bars you will filter down the reports to show only what you have clicked on to select.

The colors on the graphs show the severity of the outcome of the reports, the light blue color means a non-severe outcome while the darker color means a severe outcome.

The Reports Listing Page

The reports listing page contains a list of the reports that match the applied filters. Clicking on the drop-down arrow to the left of every report will open up that report's options. Once there, the user can view and annotate that specific report's narrative. The user can also send the report to any of the cases or to the trash bin.

The top of the reports listing page contains the user's cases. There, the user has the option to create new cases or to look at only the reports contained in a specific case.

Pressing the button on the bottom right will bring up the case summary side panel, which has more details about all the cases.

User Dashboard Page

This page has more in-depth information about the user's cases. The user can also set cases as "inactive" if they no longer wish to make changes to that case.

Appendix D: User Study Tasks

Script for investigators:
[Read over the consent form and have the participants sign the form]
[Sign the consent form]
[Read over the introduction paragraph that explains the system]
[Start the survey]
[Guide participants as needed if they cannot find or understand any of the steps in the user study]

[Participants will be observed as they complete the tasks. The proctors will record]

- The time for how long it takes the participants to complete each task.
- The answer they provide to the questions.
- The level of guidance required from the proctor.
- State of the screen at the end of each task

User Study Tasks: Part 1, Familiarizing with the Filtering Methods

1. Navigate to the MEV visualization page at mev.wpi.edu:3001.
2. Log into the system using your email address.
3. Using the timeline visualization, select the month of data from **1/15/2017 - 2/12/2017.**
   - What general trends do you see in this month of reports?
4. Now select the week of data from **1/29/2017 - 2/04/2017**.
   - Estimate the proportion of reports were **severe** during the week?
   - Which **drug(s)** had the most number of reports?
   - Which **medical error** (ME) has the most severe outcomes?
5. On the treemap visualization, select any **one drug** and any **two medication error types**.
6. Using the demographics visualizations, select the **gender** with the most **severe** reports.
   - What age group has the most total reports? How many?

[Ask participants questions related to Task 1]

User Study Tasks: Part 2, Creating and adding to a case

1. Navigate to visualization page.
2. Clear all the filters
3. Filter the reports down to **2/26/2017 - 3/04/2017**.

4. Filter to reports from the **United States** that contain **'CPD'** drug
   - What is the cause with the most severe reports?
   - Which age group has the most severe outcomes?
5. Navigate to the reports listing page.
6. Create a new case named **CPD**.
7. Add 5 reports to your new case and 5 reports to the trash

[Ask participants questions related to Task 2]

User Study Tasks: Part 3, Annotating a report

1. Navigate to any report's text annotation page.
2. Highlight different text in three different colors.
3. Close the text annotation page, add the report to your **CPD** case and navigate to the user dashboard.
4. View the case summary of your **CPD** case.
   - What is the most highlighted narrative keyword type in the case?

[Ask participants questions related to Task 3]

User Study Tasks: Part 4, Finding a specific report

1. Navigate to the Visualizations page.
2. Clear all filters
3. Use the filters to find a report where a **75** year old **female** patient has reported an error with **Perrigo Hydroquinone** during **Administration** stage with a cause of **Product Quality Issue** on **12/27/2016**.
4. View this exact report inside of the reports listing page.
5. Highlight all mentions of a drug in the narrative.

[Ask participants questions related to Task 4]

User Study Tasks: Part 5, Case Management

1. Log in with the email 'mev.casestudy@wpi.edu'
2. Find the case with highest number of reports as supporting evidence?
   - What are 3 annotations from analysts on reports under the case **Advil**?

[Ask participants questions related to Task 5]

Appendix E: User Study Post-task Questions

User Study Tasks: Part 1, Familiarizing with the Build Tool Components

1. How clear is the purpose and function of the Medication Error Visualizer (MEV)?
- Extremely clear
- Very clear
- Moderately clear
- Slightly clear
- Not at all clear

2. How challenging was it to navigate to the visualizations page?
- Extremely challenging
- Very challenging
- Moderately challenging
- Slightly challenging
- Not at all challenging

3. How intuitive was it to filter down the reports by demographics (age and sex)?
- Extremely intuitive
- Very intuitive
- Moderately intuitive
- Slightly intuitive
- Not at all intuitive

4. How intuitive was it to select a drug and medication error type in the treemap visualization?
- Extremely intuitive
- Very intuitive
- Moderately intuitive
- Slightly intuitive
- Not at all intuitive

5. Did you have any problems applying the filters provided?
- Yes, I was not sure of how to apply the filters
- Yes, the visualization was slow

- Yes, the visualization did not apply the expected filters
- No problems

User Study Tasks: Part 2, Creating and adding to a case

1. How challenging was it to filter down reports a second time?
- Extremely challenging
- Very challenging
- Moderately challenging
- Slightly challenging
- Not at all challenging

2. How intuitive was it to create a new case?
- Extremely intuitive
- Very intuitive
- Moderately intuitive
- Slightly intuitive
- Not at all intuitive

3. How intuitive was it to add reports to your new case and to the trash?
- Extremely intuitive
- Very intuitive
- Moderately intuitive
- Slightly intuitive
- Not at all intuitive

User Study Tasks: Part 3, Annotating a report

1. How intuitive is the navigation to annotate a report?
- Extremely intuitive
- Very intuitive
- Moderately intuitive
- Slightly intuitive
- Not at all intuitive

2. How challenging was it to annotate the report?
- Extremely challenging
- Very challenging

- Moderately challenging
- Slightly challenging
- Not at all challenging

User Study Tasks: Part 4, Find a specific report

1. Did the UI help you find the report you were looking for?

- Yes
- No, the UI was difficult to use
- Optional: Elaborate more on why or why not

2. How challenging was it to find the report?
- Extremely challenging
- Very challenging
- Moderately challenging
- Slightly challenging
- Not at all challenging

User Study Tasks: Part 5, Case Management

1. How intuitive was it to find the case with the most supporting reports?
- Extremely intuitive
- Very intuitive
- Moderately intuitive
- Slightly intuitive
- Not at all intuitive

2. Do you have any additional comments on your overall experience with the Medication Error Visualizer (MEV)?

- [open ended]

Appendix F: Application Help Page

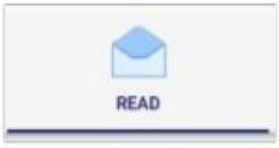
Sign In Page

This is the sign-in interface. Here the user will enter a username that they will use to analyze the FDA data.

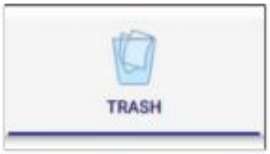
If there is no user already in the system with the user input, a new user will be created with two predefined cases (**Read** and **Trash**).

User Dashboard

This is the user dashboard interface. This is where you will be directed after you have logged in. From here you will see the two predefined cases **Read** and **Trash.** These are two cases that will come in handy for sorting through the reports your have narrowed down to build your cases.



Read: This is a pre-generated case to store the reports that you want to mark as being already read. Reports in this case will display as grey on the report listing page



Trash: This is a pre-generated case to store the reports that you do not want to show up the report listing page

Go To Visualization: This button will bring the user to the visualization page. The visualization is where you will work to narrow down reports and analyze trends.

Go To Reports Listing: This button will bring the user to the reports listing page where the user will be presented with either a broad list of reports or if you have already visited and used the visualization page, it will presented with a list of reports that show with your applied filters.

Visualizations

This is the data visualization interface. Here you will find the most recent two weeks of data in the database. This will default load 48,050 reports which can be filtered down using any of the visible filters in the interface. Below you will find the components explained further.

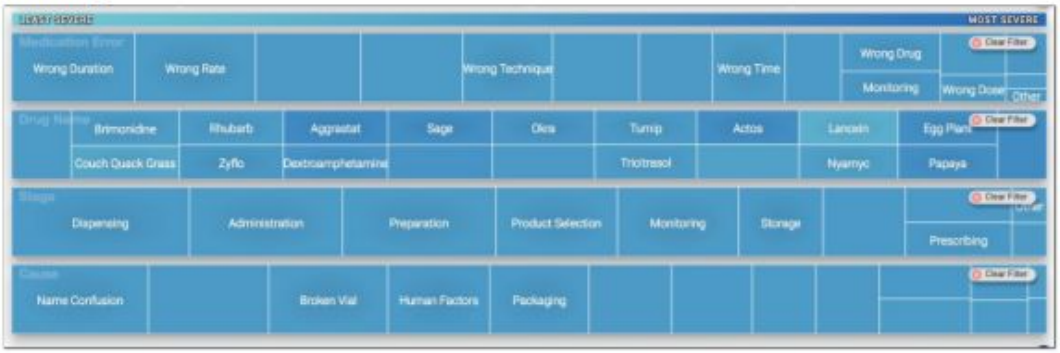
Go to Reports Listing: This button will bring the user to the reports listing page. Which will list all the queried reports which attributes are being displayed in the visualization interface.

Timeline: The timeline will show the number of reports in frequency as well as the correct severity percentage based on the number of severe to non-severe reports.

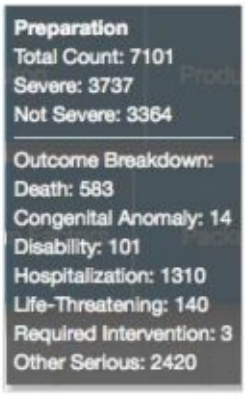
Controls

- **Ctrl + Scroll:** Using these two inputs will allow you **zoom** in and out on the timeline to decrease or increase the date range from which you can select from.

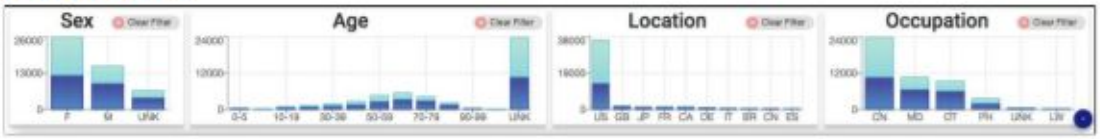
Treemap



Here you can see the treemap section of the visualization interface. This is separated into five sections, the legend, medication error treemap, drug name treemap, stage treemap and the cause treemap. The four treemaps are all selectable by each of the present report attributes in its categories. These treemaps will be the fastest way to narrow down your report search. The boxes that display with no text, can still be read if you **hover** your mouse over the treemap, and it will display a tooltip as such:



Demographics

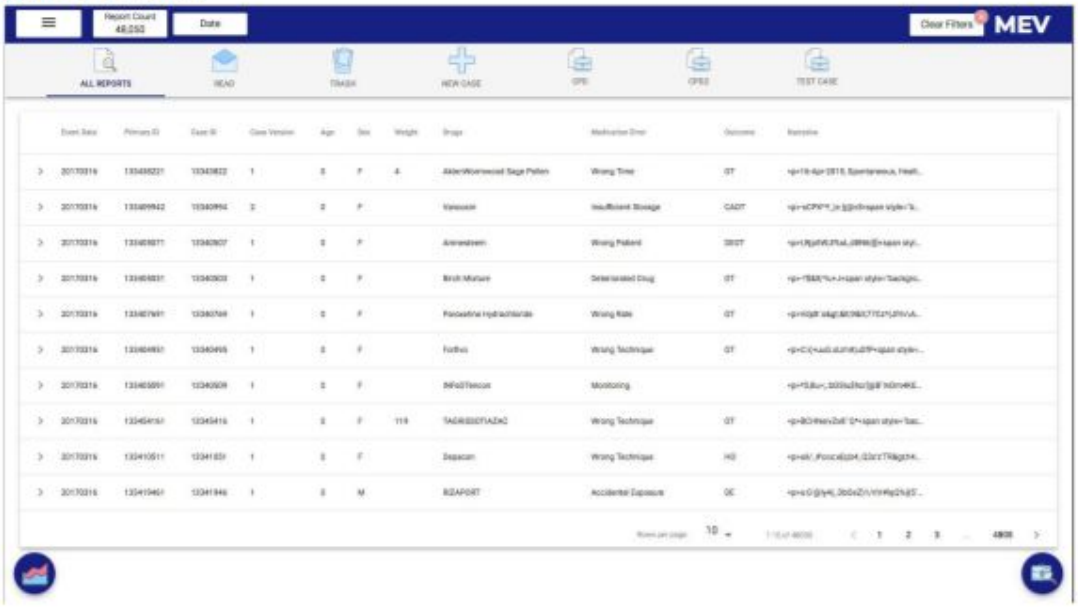


This section of the interface will show the queried reports demographic data. These graphs will also show the severe to non-severe reports that are appearing in your reports query.

Controls:

- **Shift + Click:** Holding shift and clicking will allow you to select multiple elements from either the **Treemap** section or the **Demographic** section of the visualizations interface. When the user released the shift key, the query will send.

Reports Listing



This is the reports listing interface where all your queried reports will be visible to the user here under **ALL REPORTS.** After the default active tab, you will see the two predefined cases, an option to create a new user defined case, and all other active cases that the user has ownership of.

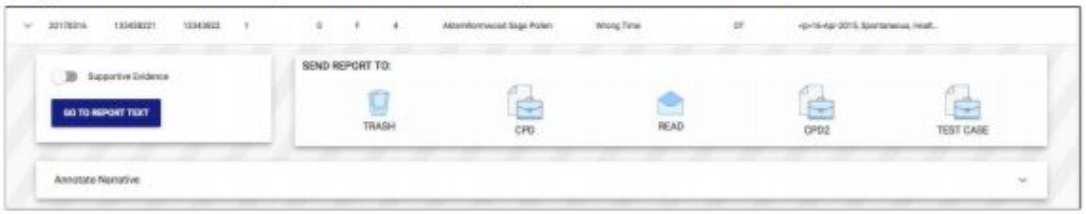
Go To Visualization: This button will bring the user to the visualization page. The visualization is where you will work to narrow down reports and analyze trends.

Open Case Summary: This button will open a case summary listing which will list all your active cases in a expandable list. It will open a drawer to the right which will look like:

In this interface you can click one of the expandable cases to see statistic that will summarize user built cases with all their report annotations. The summaries may appear like so:

Report Item

Inside the report listing you can click and expand on the report to add to cases based on **Supportive** or **Primary Evidence**. The expandable menu can be seen below:



Here is where you can add a report to a case. Whether you would like to add a report as supportive or primary evidence. Once you have toggled the report to the desired evidence type, you can click which case you would like to add the report too with that specified evidence type.

Inside the report item view, you can see another expandable view which will allow the user to highlight and annotate the report to help build cases which will be displayed in the case summary view. The report annotation is displayed as:

You will be able to highlight sections of text and select the attribute that you would like to highlight to show importance throughout the report.

Appendix G: User Study Data Spreadsheet

				Task 1
~	50-50%	Silk, Malt	Wrong Rate	~
During Task Questions				
Q1	Q2	Q3	Q4	Q5
No data on weekends, 5 days of the week it goes up	Little less than ha	Cockroach	Wrong Duration	70-79, 1
Old people died more, more women with less severe, US has more deaths	About 40%	Silk, Malt	Wrong Rate	60-69, 2863
Severe outcomes stay constant, non-severe outcomes are increasing	40%	Silk	Wrong Rate	Tie between 20-2
More females, avg age around 65	About 50%	Silk, Malt	Wrong Rate	20-29, 50-59 at 1
Uniform severity	50%		Stage	50-59, 2
Lots more females than males, looks like primarily 50-80, normal dist, all US	35%	Silk, Malt	Wrong Rate	40-49, 2
More females, more reports with 60-60 and unkown, and US is primary location	33%	Silk, Malt	Wrong Rate	tie, 60-69 with 3 r
Not alot of reports on the weekend	40%	Silk, Malt	Wrong Rate	Female, 40-49
most age 60-69, more female, most consumer, all U.S. based	Little under half	Silk, Malt	Wrong Rate	40-49, 60-69, 80-
every 7 days something is happening	about 45-55%	Silk, Malt	Wrong Rate	10-19, UNK, 1 re
A lot of US reports	Half	Silk	Name confusion	40-49
Weekends we get no reports	40%	Pennisaid	Wrong Rate	Female, 40-49, U
All of the colors were about the same	~50%	Silk, Malt	Wrong Rate	Female, 50-59
more females	~25%	Silk, Malt	Wrong Rate	Female, 20-29, 6
More females, but more not sever then severe	50%	Silk, Malt	Wrong Rate	Male and Female
Lots of consumers, same amount of issues that are severe	40%	Silk, Malt	Wrong Rate	UNK, 1
No reports on weekends	45%	Silk, Malt	Wrong Rate	Female, 50-59
Lots of US reports	50%	Dyanavel XR	Other	Male, UNK

Task 1				
~	~	~	~	~
After Task Questions				
Q1	Q2	Q3	Q4	Q5
3	2	4	1	NO
2	5	4	2	
4	2	5	3	
4	1	4	2	No, but the multiple selection bug
4	2	5	2	Cannot select multiple filters
4	1	4	3	Yes, trouble with timeline
4	3	1	1	Yes, trouble with timeline
5	3	1	2	Yes, with the timeline
5	1	5	5	No
4	2	4	4	Yes, not sure of how to apply the filters
4	3	3	4	Yes, did not know to press "Set Date"
5	5	5	3	Yes, it did not apply the expected filters
5	1	4	3	Yes, was not sure of how to apply the filters
1	5	2	2	No
3	5	2	3	Yes, the date range
4	1	3	4	No
4	1	4	3	No
3	3	3	3	Yes, selecting multiple drugs

Task 2				
Name Confusion, Human Factors	**UNK**	~	~	~
During Task Questions		After Task Questions		
Q1	Q2	Q1	Q2	Q3
Name Confusion, Human Factors	40-49	3	4	1
Name Confusion, Human Factors	40-49	3	3	4
Data entry error	Unkown	3	5	5
Name Confusion, Human Factors	40-49	1	5	2
Human Factors	UNK	5	5	5
Name Confusion, Human Factors	40-49	2	5	3
Human Factors	40-49, unk	4	1	3
Human Factors	40-49	4	3	3
Name Confusion, Human Factors	40-49	1	5	5
Wrong Patient, Wrong technique	UNK	4	1	3
Human Factors	40-49	4	5	4
Human Factors	40-49	1	5	4
Name Confusion, Human Factors	UNK	4	5	4
Name Confusion, Human Factors	40-49, unk	4	2	3
Name Confusion, Human Factors	40-49	5	1	4
Name Confusion, Human Factors	40-49	1	4	4
Name Confusion, Human Factors	UNK	1	5	3
Human Factors	UNK	2	2	1

Task 3		
~	~	~
During Task Questions	After Task Questions	
Q1	Q1	Q2
Dosage and weight	1	3
Reaction	3	2
	5	5
Weight	2	1
Drug	5	5
Drug, Age	2	4
Age	2	5
Age	2	4
Wieght	5	1
Drug and Dosage	4	4
Dosage	4	4
Drug, Dosage, Weight, Indication	5	1
Drug, Dosage, Weight, Age	5	2
Drug	2	4
Age, Gender, Indication	2	1
Drug, Reaction	4	1
Dosage	4	2
Age	3	1

Task 4	
~	~
After Task Questions	
Q1	Q2
No, found it but was difficult	5
No, couldn;t find it	5
Not there	5
Found it, took time, difficult	3
Hard to find exact drug	3
Somewhat, hard to find name of drug because nothing shows on treemap	3
No	3
Yes	4
Yes	1
Ses and Age were easy, when you dont see the drug in the top 20 it is difficult	3
Yes	3
Yes, but wished there was an autocomplete for selecting the drug	2
Yes, the timeline is hard to use	1
Yes	4
Yes	3
Yes	1
Yes, Timeline is hard to use	3
No	5

Task 5	
Advil and Aspirin	~
During Task Questions	After Task Questions
Q1	Q1
	2
Advil and Aspirin	3
Tylenol and Aspirin	1
Advil and Aspirin	3
Advil and Aspirin	2
Advil and Aspirin	2
Advil and Aspirin	5
Advil and Aspirin	5
Advil and Aspirin	3
Advil and Aspirin	1
Advil and Aspirin	1
Advil and Aspirin	4
Advil and Aspirin	2
Tylenol and Aspirin	3
Advil and Aspirin	2
Advil and Aspirin	3

Appendix H: User Study Application Comments

Timeline is hard, scrolling should move in the opposite directions that it does, outcome breakdown unclear, weird to have to hit the (x) in the corner of buttons, thought treemap went off screen bc no words, Set date is hard to see, messed up the first task, Opening and closing each report sucks, Save button is weird, didn't realize report narrative was on another page, If you have everything open in the report listing page it is hard to see things, View Case Summary was hard to navigate to, Things scroll off the screen which is bad.

Sign in instead of create user, No ratio # in tooltip, Timeline is super hard to use, Tooltip for top bar dont work (cant tell what filtering by), Case creation unclear, individually adding cases is bad, Unclear if text saved (add save completed or something).

Trouble figuring out that you need to click and drag on the timeline.

Hard to reset date, Hard to tell differences between boxes, If timeline is too zoomed out, you skip dates, when tried to add reports to case sometimes it didn't go until clicked multiple, report listing page was easiest to use viz page hardest.

Hard to use timeline, would like to be able to select highlight color then highlight instead of selecting then pressing button.

timeline controls are bad, timeline is slow, timeline resets on scroll, checkboxes instead of individual moves, annotate narrative should be another color than white not obv, only clicking on arrow is bad.

Application would be good workflow if trained on the system before.

No, I am not a computer person.

Impossible to tell the colors apart on treemap, legend not helpful, in vis not severe does not display 0, Visual confirmation for moving a report to the trash.

Shift key didn't work, names were too long and don't show on treemap, didn't use clear all button, adding report to a case is slow, wanted to be able to just click the annotation color then highlight, Hard time to understand the difference between a report and a case, if the case had its own tab from the navigation that would be nice.

Had a hard time finding the case summary, did not know to press "Set Date" button.

Not a clear understanding of difference between a case and a report, Toggle was unclear for primary/supportive, Clear filters should be a full size button.

Clear button in annotation should be renamed to Erase, Wish we could click the button then highlight the text, Wished there was a checkmark to move many reports, Clear filter button should be an actual button, Unsure where to find how to see the count of supportive reports in a case, Why did the filters stay when i logged out.

No it looks good.

I don't like how the names don't fit in the boxes... thought they were blank.

Looks good, likes the colors.

Timeline scrolling was backwards, wish you could select a color before highlighting text in annotation.

Not sure what the middle graphs mean (treemap), too many reports all at once.

Appendix I: GitHub ReadMe and Redux Tutorial

MEV-MQP

Setup

Dependencies

- NodeJS & NPM (download the most recent version for your OS at https://nodejs.org/en/download/)
- Postgres (download the most recent version for your OS at https://www.postgresql.org/download/)
- (OPTIONAL mac/linux only) Redis (download the most recent version for your OS at https://redis.io/download) ** Follow the instructions at (http://rejson.io/#building-and-loading-the-module) to set up the json module for redis

Database

Set-up PostgreSQL DB

1. Run the command ' `createdb faers` ' in your command line interface
2. Download our DB dump file from the Google Drive called latest.sql
3. Run the command ' `psql faers < latest.sql` ' to import data into the database (this may take some time)

Dump PostgreSQL DB

1. run the command ' `pg_dump -h localhost -U mevuser -W -d faers > latest.sql` ' in the postgres command line interface

Local Startup

1. Navigate to outermost folder in the repository on your computer
2. Run `npm install`
3. Navigate to front-end folder
4. Run `npm install`
5. Navigate to back-end folder
6. Run `npm install`
7. Navigate to outermost `mev-mqp` folder
8. Run `npm start` to start the application

Remote Startup

1. SSH into the `mev.wpi.edu` machine and navigate to `/MEV/back-end`
2. It is recommended to run the server in a linux **screen** session
   - If needed, create a new screen by typing `screen`
   - Use `screen -list` to find the name of the screen
   - Re-attach to a screen with `screen -r name`
   - Detach from a screen by pressing `ctrl + a` followed by `d`
3. If the node server is running you can stop it with `ctrl-c`
4. Start the server by running `npm run startBoth` . This starts the node server and the redis cache
5. Detach from the screen by pressing `ctrl + a` followed by `d`

Remote Deploy

1. SSH into the `mev.wpi.edu` machine and stop the node server with `ctrl-c`

2. Inside of the `mev-mqp` folder run the `deploy.sh` file as '`sh deploy.sh username`' where username is your user account name on the `mev.wpi.edu` machine.
    - This will build the React app into a single build folder *-this should not be pushed to GitHub-*
    - Copy this folder and the app.js to your user on the remote server
    - Delete the previous build folder and app.js on the remote server
    - Move the files from your user to the `/MEV/back-end` directory
3. This will require you to enter your `mev.wpi.edu` system account a few times.
4. SSH into the `mev.wpi.edu` machine to start the node server again.

Project Layout

```
|-back-end
|-front-end
|___public
|___src
| |___resources
| |___components
| | |___visualization
| | | |___components
| | | | |___demographics
| | | | | |___components
| | | | | | |___components
| | | | |___timeline
| | | | | |___components
| | | | | | |___components
| | | | |___treeMap
| | | | | |___components
| | |___portal
| | | |___images
| | | |___userComponents
| | |___cases
| | |___components
| | |___editor
| | | |___images
| | |___reports
| | | |___components
| |___actions
| |___reducers
```

Working on the Front End

To make changes to the UI and other aspects of the front end, you must edit assets found in the front-end folder.

Working with React + Redux

When editing the global state that Redux manages, you need to make changes in several different locations.

Useful Readings/Videos for Learning React and Redux

- Thinking In React
- State and Lifecycle
- React Component Lifecycle
- JSX in Depth

- Helpful Youtube Channel
- React and Redux Rapid Course Video Series

Outgoing Web Fetch Calls

All asynchronous http requests are done from the `/actions` folder. This is where we talk to the back-end server to get data from our database. These async calls are created using the `fetch()` function which returns a `Promise`. Please read more about javascript promises here.

Adding to the Redux (Global) State

When adding to the global state you must create a function inside of a file in the `/actions` folder, add a case inside of a file in the `/reducers` folder and add an import to the file you are adding to the Redux state from.

In the `timelineActions.js` file we have:

```
export const setTimelineMinimizedToggle = toggle =>
  dispatch => dispatch({ type: 'TOGGLE_TIMELINE_MINIMIZED', timelineMinimized: toggle });
```

In this example we are adding a value `toggle`, labeled as `timelineMinimized` with a type `TOGGLE_TIMELINE_MINIMIZED` this is just a string to know what to listen for in the Reducer. This `dispatch()` function is what we need to call in order to have this information be sent to the reducers and be added to the global state.

In the `timelineReducer.js` file we have:

```
export default (state = initialTimelineState, action = {}) => {
  switch (action.type) {
    case 'SET_ENTIRE_TIMELINE':
      return Object.assign({}, state, { entireTimelineData: action.entireTimelineData });
    case 'TOGGLE_TIMELINE_MINIMIZED':
      return Object.assign({}, state, { timelineMinimized: action.timelineMinimized });
    default: return state;
  }
};
```

We are listening for the same `TOGGLE_TIMELINE_MINIMIZED` type, when we find that type we are setting the state to have the `timelineMinimized` value passed from the actions.

In the `Timeline.jsx` file we use this action like such:

We import the functions from the actions.

```
import { setTimelineMinimizedToggle, getEntireTimeline, setSelectedDate } from '../../../../actions/timelir
```

We then connect these actions to the current component as `props`.

```
export default connect(
  mapStateToProps, // Used for reading the Global state
  { setTimelineMinimizedToggle, getEntireTimeline, setSelectedDate },
)(withStyles(styles)(Timeline));
```

We add these functions to the proptypes of our component.

```
static propTypes = {
  getEntireTimeline: PropTypes.func.isRequired,
  setSelectedDate: PropTypes.func.isRequired,
  setTimelineMinimizedToggle: PropTypes.func.isRequired,
}
```

We can call these functions from our props like this.

```
this.props.setTimelineMinimizedToggle(true);
```

Getting from the Redux (Global) State

To get from the global state, we need to check to make sure the information is in the state properly and then import and connect it into our component.

First make sure we have the proper `case` clause for the information we are looking for. From the example above, we are looking for `TOGGLE_TIMELINE_MINIMIZED` inside of the `timelineReducer.js` .

```
export default (state = initialTimelineState, action = {}) => {
  switch (action.type) {
    case 'SET_ENTIRE_TIMELINE':
      return Object.assign({}, state, { entireTimelineData: action.entireTimelineData });
    case 'TOGGLE_TIMELINE_MINIMIZED': // Here it is
      return Object.assign({}, state, { timelineMinimized: action.timelineMinimized });
    default: return state;
  }
};
```

We also need to make sure we have imported the `timelineReducer` into the index reducer inside of the `/reducers/index.js` file.

```
import demographic from './demographicReducer';
import filters from './filterReducer';
import multiSelectFilters from './multiSelectFilterReducer';
import user from './userReducer';
import mainVisualization from './visualizationReducer';
import timeline from './timelineReducer';

/**
 * Redux Reducer that combines all of the other reducers to build the Redux State
 */
export default {
  demographic,
  filters,
  multiSelectFilters,
  mainVisualization,
  timeline, // Exported as 'timeline'
  user,
};
```

We can see we import the `timelineReducer` and export it with the name `timeline` . **This name is important.**

Now we can go to our component `Timeline.jsx` and import the global state.

```
const mapStateToProps = state => ({
  entireTimelineData: state.timeline.entireTimelineData,
  demographicSexData: state.demographic.sex,
});

export default connect(
  mapStateToProps,
  { setTimelineMinimizedToggle, getEntireTimeline, setSelectedDate },
)(withStyles(styles)(Timeline));
```

Here in the `mapStateToProps` function we are getting from the global state with:

```
entireTimelineData: state.timeline.entireTimelineData,
```

It is `state.timeline` since that is what we exported as inside of the `reducers/index.js` file. And in this case we are getting the `entireTimelineData` from above:

```
case 'SET_ENTIRE_TIMELINE':
  return Object.assign({}, state, { entireTimelineData: action.entireTimelineData });
```

Inside of our component we need to add this to the proptypes.

```
static propTypes = {
  entireTimelineData: PropTypes.arrayOf(
    PropTypes.shape({
      init_fda_dt: PropTypes.string.isRequired,
      serious: PropTypes.number.isRequired,
      not_serious: PropTypes.number.isRequired,
    }),
  ).isRequired,
}
```

This data can now be accessed from the props as such:

```
this.props.entireTimelineData
```

Working on the Visualization page

Assets for the visualization page exist within the visualization folder src/components/visualization.
visualization/App.js uses the components for the treemaps, demographics, and timeline which assets are each in their respective folder inside visualization/components.

Working on the Reports Listing page

Assets for the reports listing page exist within the reports folder src/components/reports.
reports/ReportsList.jsx uses the components for the report listing grid which assets are all contained in the folder reports/components.

Working on the Narrative Editor page

All assets for the narrative editor page exist within the editor folder src/components/editor

Working on the Login page, About page, or Dashboard page

All assets for these pages exist within the portal folder src/components/portal
Assets for the dashboard page exist inside the portal/userComponents folder.

Working on the Back End

Our backend is a one file server that receives requests sent from the front end located at back-end/App.js

Adding and modifying endpoints

We use the express library (https://expressjs.com/) for our backend and documentation for use can be found at https://expressjs.com/en/4x/api.html#app.