### **Overview of Application Requirements and Workflow**

1. **Data Display**:
   * The main task involves selecting specific fields from a table to display relevant data.
   * The objective is to ensure that the displayed data is concise and directly sourced from a single table, which simplifies the data retrieval and presentation process.
2. **Export Functionality**:
   * There was a discussion on whether to create a separate export page or simply add an export option as a column on the existing page.
   * The consensus was to avoid a separate export page. Instead, the export functionality could be integrated directly into the existing page, which would streamline the workflow and avoid redundancy.
3. **Server Data Review**:
   * It was suggested to review the data by pulling up a recent batch run on the server. This allows the team to better understand the data structure and what fields might change in future runs.
   * This process is crucial because the fields in the table are dynamic and subject to change with different batch runs, making it necessary to continuously adapt the table structure.
4. **Field Changes and Data Runs**:
   * Some fields will change with future data runs, which is why the table hasn’t been finalized yet.
   * The team discussed the need to review the current batch run data to better anticipate changes and finalize the table structure.
5. **Expertise and Collaboration**:
   * Sonata has gained considerable knowledge about R programming, though still in the learning phase.
   * Liz is recognized as an expert in R, providing valuable guidance to others, while also acknowledging the challenges of learning advanced programming concepts like nested dictionaries in Python.
6. **Challenges with Data Structure**:
   * There was a light-hearted exchange about the complexity of learning programming, particularly the difficulties associated with understanding and managing nested dictionaries in Python.
7. **Batch Run Example**:
   * The conversation shifted to reviewing a specific batch run, which involved running approximately 110 cases. Each case may result in multiple individual records, depending on the number of individuals associated with each case.
   * The team is working on summarizing this data in a single table, which can be challenging given the variability of the data.
8. **Data Uploads and Case Management**:
   * The discussion highlighted the complexity of managing and uploading case data, especially when a single case might involve multiple individuals, each requiring separate documentation.
   * The team emphasized the need to track data at the individual victim level, rather than combining multiple individuals from a single case into one record.
   * They also discussed the challenges faced when uploading data from different sources, such as medical examiners and law enforcement, especially when these entities bundle or separate reports differently.
9. **Handling of Complex Cases**:
   * There was a discussion on how to handle cases where multiple locations and individuals are involved, such as a case where a person commits multiple crimes at different locations.
   * The team recognized the need to treat each individual involved as a separate record to ensure accurate data tracking and reporting.
10. **Data Extraction and Reporting**:

* The team plans to extract data from multiple sources, such as vital records, death records, medical examiners, and law enforcement reports.
* The challenge is to accurately match records to the correct individual, especially when different sources provide information in varying formats.

1. **Multiple Report Types**:

* Not all cases will have the same types of reports, but there could be multiple reports for a single case, such as law enforcement reports, medical examiner reports, and coroner reports.
* The team needs to ensure that the application can handle these varying inputs and correctly associate them with the relevant individuals.

1. **Victim-Focused Data Collection**:

* The primary focus of the data collection is on the victim, including details about their life, the suspect, and the circumstances of the incident.
* Abstractors are tasked with sifting through large amounts of data, often filled with irrelevant information, to extract what is necessary for the case.

1. **National Surveillance System**:

* The data collected is input into a front-end application developed by the CDC, as part of a national surveillance system used by all states.
* This system is crucial for standardizing data across different jurisdictions, despite the varying formats in which the data is originally collected.

1. **Workload and Funding Challenges**:

* The team is handling a significant workload with limited resources. For example, in 2023 alone, there were nearly 3,000 violent deaths in Georgia, each requiring detailed data collection and reporting.
* The funding for the program has remained flat despite increasing costs and workloads, making it difficult to expand the team or resources.

1. **Grant Funding**:

* The conversation ended with a mention of the importance of grant funding, which is essential for supporting the program’s continued operation and potentially expanding its capabilities.

This elaboration breaks down the original conversation into key points, making it easier to understand the requirements and the context of the application development.

4o

##### **You said:**

Next steps?

#### **ChatGPT said:**

ChatGPT

### **Next Steps**

1. **Data Review and Finalization**:
   * **Review Recent Data Runs**: Pull and analyze the most recent batch run data to identify any changes in the fields. This review will help in understanding the variability of the data and the fields that need to be included in the final table structure.
   * **Finalize Table Structure**: Based on the data review, finalize the structure of the table that will be used for displaying and exporting data. Ensure that the table can handle dynamic fields and future data runs.
2. **Integration of Export Functionality**:
   * **Design Export Feature**: Design the export functionality to be integrated directly into the existing page. Determine the format (CSV, Excel, etc.) and the specific fields that need to be included in the export.
   * **Implementation**: Develop and test the export functionality, ensuring it works seamlessly with the existing data display page.
3. **Collaboration and Knowledge Sharing**:
   * **Leverage Expertise**: Continue to leverage Liz’s expertise in R programming to assist with complex programming tasks, such as handling nested dictionaries in Python or other advanced data processing needs.
   * **Learning and Development**: Support Sonata in her learning journey by providing resources and mentorship in R and Python programming. This will help in handling future challenges more independently.
4. **Data Handling and Management**:
   * **Standardize Case Data Input**: Work on creating a standardized approach to managing and uploading case data, particularly for cases involving multiple individuals or multiple types of reports. Ensure that each individual record is accurately represented in the data.
   * **Develop Protocols for Complex Cases**: Establish protocols for handling complex cases where multiple locations, individuals, and report types are involved. This will help in maintaining data accuracy and consistency.
5. **Application Enhancements**:
   * **Refine Front-End Application**: Work on refining the front-end application to better support the data collection and reporting needs. This may include improving the user interface for abstractors, making it easier to input and manage data.
   * **Ensure Compliance with CDC Standards**: Ensure that all data collected and processed through the application complies with CDC’s national surveillance system standards. This will involve continuous testing and validation of the data input process.
6. **Resource and Funding Management**:
   * **Optimize Workload Management**: Given the significant workload and flat funding, explore ways to optimize the team’s efficiency. This may involve automating certain tasks or streamlining the data collection process.
   * **Seek Additional Funding**: Identify opportunities for additional grant funding to support the program’s expansion. This could involve writing grant proposals or seeking partnerships with other organizations.
7. **Documentation and Training**:
   * **Document Processes**: Create detailed documentation of all processes, including data review, table structure finalization, and export functionality. This will be useful for onboarding new team members and for future reference.
   * **Conduct Training Sessions**: Organize training sessions for team members on new features, tools, and processes, ensuring everyone is aligned and capable of performing their tasks efficiently.
8. **Monitoring and Feedback**:
   * **Implement Feedback Loop**: Establish a feedback loop where team members can regularly report any issues or suggest improvements for the application and data management processes.
   * **Monitor Data Accuracy**: Continuously monitor the accuracy of the data being collected and reported, making adjustments as necessary to ensure the highest quality of data.

By following these next steps, the team can move forward efficiently with the project, ensuring that all aspects of the application and data management processes are well-coordinated and aligned with the overall goals.

The application you're working on is designed to manage, process, and display complex data related to cases, particularly focusing on data collection, export, and reporting functionalities. Here's a breakdown of what the application is about:

### **\*\*1. Purpose and Use Case**

* **Data Collection and Management:** The primary goal of this application is to collect, manage, and display detailed data related to cases, which might involve multiple individuals and varying types of reports (e.g., law enforcement reports, medical examiner reports).
* **National Surveillance System:** The data collected through this application will be used as part of a national surveillance system managed by the CDC (Centers for Disease Control and Prevention). This system aims to standardize data collection across different jurisdictions, ensuring consistent and accurate reporting.

### **\*\*2. Core Functionality**

* **Data Display:** The application is responsible for selecting and displaying specific fields from a central table that contains case data. The displayed data must be concise, directly sourced from the table, and relevant to the users' needs.
* **Export Functionality:** Users can export the displayed data in formats like CSV or Excel. This allows them to create reports or share data with other stakeholders without needing a separate export page. The export feature is integrated into the existing data display page, simplifying the workflow.
* **Data Review:** The application allows users to pull and review recent batch runs from the server. This is crucial for understanding the data structure, especially since the fields in the table may change with different batch runs.
* **Handling Dynamic Data:** Some fields in the data table will change with future data runs, making it necessary for the application to adapt and update its structure dynamically.
* **Case Management:** The application handles complex cases where multiple individuals or multiple reports are associated with a single case. It tracks data at the individual level, ensuring that each person involved in a case is accurately represented in the data.

### **\*\*3. Challenges Addressed**

* **Complex Data Structures:** The application deals with complex data structures, including nested dictionaries and dynamic fields that can change with different data runs.
* **Multiple Data Sources:** The application must integrate data from various sources, such as vital records, death records, medical examiners, and law enforcement reports, and ensure that the information is matched accurately to the correct individuals.
* **Compliance with National Standards:** All data collected and processed through the application must comply with the CDC's national surveillance system standards, ensuring that the data is standardized and can be used for nationwide reporting and analysis.

### **\*\*4. \*\*Collaboration and Expertise**

* **Teamwork:** The development of this application involves collaboration among team members with different levels of expertise. For instance, Liz, an expert in R programming, provides guidance on complex programming tasks, while Sonata is in the learning phase, gaining knowledge in R and Python.
* **Continuous Learning:** The team emphasizes continuous learning and development, supporting each other in overcoming challenges related to data structure, programming, and application design.

### **\*\*5. Application Context**

* **Public Health and Safety:** The application is part of a broader public health initiative to track and report on violent deaths and other critical incidents. The data collected is vital for understanding trends, informing policy decisions, and improving public safety.
* **Resource Management:** The application is developed in an environment with limited resources and funding, making it essential to optimize the workflow, manage workloads efficiently, and seek additional funding opportunities to support the program's growth.

### **\*\*6. Future Enhancements**

* **Refining the User Interface:** The application may undergo enhancements to improve the user interface, making it easier for abstractors (individuals who enter and manage the data) to input and manage data.
* **Expanding Functionality:** As the application evolves, it may incorporate additional features such as automated data validation, enhanced reporting capabilities, and more robust data handling protocols for complex cases.

In summary, this application is a critical tool for managing, processing, and reporting case data in a standardized and efficient manner, supporting a national public health initiative led by the CDC. It addresses the challenges of handling complex data, ensuring data accuracy, and optimizing resources in a demanding environment.

Optimizing data in the context of application development involves improving the efficiency of data storage, retrieval, processing, and management to enhance performance, reduce costs, and ensure data quality. Here are several strategies to optimize data in your application:

### **1. Database Optimization**

* **Indexing:**
  + **Create Indexes:** Identify the most frequently queried fields in your database tables and create indexes on them. Indexes speed up read operations by allowing the database to find data without scanning the entire table.
  + **Use Composite Indexes:** When queries often filter on multiple columns, create composite indexes to cover those queries. However, be cautious with too many indexes as they can slow down write operations.
* **Normalization vs. Denormalization:**
  + **Normalization:** Organize data to reduce redundancy and improve data integrity. This involves structuring data into related tables and using foreign keys.
  + **Denormalization:** In some cases, denormalization is better for performance, especially for read-heavy applications. This involves merging tables or adding redundant data to reduce the number of joins.
* **Partitioning:**
  + **Horizontal Partitioning:** Split large tables into smaller, more manageable pieces called partitions. Each partition can be stored on different disks or servers, improving query performance.
  + **Vertical Partitioning:** Split a table by columns, placing frequently accessed columns in one table and less accessed columns in another, reducing the amount of data read during queries.
* **Query Optimization:**
  + **Analyze and Optimize Queries:** Use query profiling tools to analyze slow queries and optimize them. Rewrite complex queries, avoid unnecessary joins, and use subqueries judiciously.
  + **Caching Results:** Cache the results of expensive queries that are frequently accessed but infrequently updated. Use Redis or Memcached to store cached data.

### **2. Data Storage Optimization**

* **Efficient Data Types:**
  + **Choose Appropriate Data Types:** Use the most efficient data types for your fields. For instance, use INT instead of BIGINT for integer fields when possible, and VARCHAR with a length limit instead of TEXT.
  + **Avoid Nulls When Possible:** Nulls can lead to inefficient storage and complex query conditions. Consider default values where appropriate.
* **Data Compression:**
  + **Database Compression:** Enable data compression on your database tables, especially for large datasets, to reduce storage space and improve I/O performance.
  + **File Compression:** Compress exported files (e.g., CSV, Excel) before storing or transferring them, reducing storage requirements and speeding up data transfer.

### **3. Data Processing Optimization**

* **Batch Processing:**
  + **Batch Inserts and Updates:** When dealing with large volumes of data, perform bulk inserts and updates instead of individual operations. This reduces the overhead and speeds up the process.
* **Asynchronous Processing:**
  + **Use Asynchronous Tasks:** Offload long-running or resource-intensive tasks (e.g., large data exports, complex calculations) to background jobs using tools like Celery, allowing the application to remain responsive.
* **Data Cleaning and Validation:**
  + **Automate Data Cleaning:** Implement automated scripts or processes to regularly clean and validate data, removing duplicates, correcting errors, and standardizing formats.
  + **Validate at Entry:** Ensure that data is validated at the point of entry to prevent invalid or corrupt data from entering the system.

### **4. Data Retrieval Optimization**

* **Pagination:**
  + **Implement Pagination:** For large datasets, implement pagination to load data in chunks rather than all at once, improving performance and user experience.
* **Lazy Loading:**
  + **Load Data on Demand:** Use lazy loading to defer the loading of data until it is needed. This is especially useful for large data structures or when dealing with related data.
* **Data Caching:**
  + **Cache Frequently Accessed Data:** Store frequently accessed data in memory using caching mechanisms (e.g., Redis, Memcached) to reduce database load and speed up retrieval.

### **5. Data Integrity and Quality Optimization**

* **Regular Audits and Backups:**
  + **Conduct Data Audits:** Regularly audit your data to ensure its integrity, accuracy, and completeness. Use automated tools to identify anomalies or inconsistencies.
  + **Automated Backups:** Implement automated backup solutions to regularly back up your data, ensuring quick recovery in case of data loss or corruption.
* **Data Consistency:**
  + **Ensure Consistency Across Systems:** If your application integrates with multiple data sources (e.g., external APIs, CDC systems), implement consistency checks to ensure data remains consistent across all systems.

### **6. Scalability and Performance Optimization**

* **Load Balancing:**
  + **Distribute Workload:** Use load balancers to distribute database queries and data processing tasks across multiple servers, ensuring even distribution of the load and improving performance.
* **Sharding:**
  + **Database Sharding:** For extremely large datasets, consider sharding the database, which involves splitting the data across multiple database instances based on a key (e.g., user ID).

### **7. Data Archiving and Retention**

* **Archive Old Data:**
  + **Move Inactive Data:** Periodically archive older, infrequently accessed data to a separate storage system, reducing the load on the primary database and improving query performance.
* **Data Retention Policies:**
  + **Implement Retention Policies:** Define and enforce data retention policies to automatically delete or archive data that is no longer needed, ensuring the database remains optimized.

By implementing these strategies, you can optimize data management within your application, ensuring it runs efficiently, scales effectively, and provides a smooth user experience while maintaining high data quality and integrity.

Based on the images and the additional details you provided, here's what I understood:

### **Image Analysis:**

1. **First Image (Table Interface):**
   * This image appears to show a table within a case abstraction tool, part of the "Georgia Violent Death Reporting System."
   * The table includes columns like:
     + **Upload Document(s):** Suggests where documents related to the case can be uploaded.
     + **PersonId, Incident Id, First Name, etc.:** Likely identifiers for the individual case being handled.
     + **Certificate Number, Key, Status, View Results:** Columns for tracking the progress and details of the case, including the ability to view results.
2. **Second Image (Detailed Case Information):**
   * This image seems to display a detailed view of a particular case, specifically focusing on:
     + **Demographics:** Information about the individual, including name, age, race, and residence.
     + **Injury Information:** Details about the incident, such as the manner of death and the location of injury.
   * The interface allows the user to navigate through various sections like "Summary," "Weapon & Suspect Information," and "Circumstances."

### **Design Plan Based on the Images:**

1. **Data Display:**
   * **Concise & Single Source:** Ensure that the table on the first image is designed to display concise information sourced from a single table, as discussed.
   * **Field Selection:** Focus on selecting fields that are most relevant to the case management process, particularly those that may change with each batch run.
2. **Export Functionality:**
   * **Integration in the Existing Page:** Rather than creating a separate export page, integrate an export option directly into the existing table interface. This could be a column where users can select to export the data in different formats.
3. **Dynamic Data Handling:**
   * **Field Changes:** Prepare the table to accommodate dynamic changes in fields that may occur with different data runs.
   * **Batch Run Review:** Regularly review recent batch runs to update the table structure accordingly.
4. **Case Management:**
   * **Individual Case Tracking:** Implement a system where each victim is treated as a separate record to manage complex cases, especially those involving multiple individuals or locations.
   * **Data Uploads:** Design the upload functionality to handle different document types from multiple sources (e.g., medical examiners, law enforcement).
5. **Reporting & Extraction:**
   * **Data Matching:** Develop a method to accurately match records from different sources, ensuring that each report is associated with the correct individual.
   * **Handling Multiple Reports:** Make sure the system can handle various report types and link them appropriately to individual cases.
6. **Victim-Focused Data Collection:**
   * **Streamlining Data Collection:** Assist abstractors by designing the interface to make it easier to sift through large amounts of data and extract only the relevant details.
   * **National Surveillance System Compliance:** Ensure that the data collected aligns with the standards required by the CDC's national surveillance system.

### **Conclusion:**

The images align with the detailed design requirements you've provided. The primary focus will be on making the system user-friendly while ensuring it can handle complex, dynamic data related to violent deaths in Georgia. By streamlining data display, export functionality, and case management, the tool can effectively support abstractors in their critical work.

# **Georgia Violent Death Reporting System: Case Abstraction Helper Tool**

## **Overview**

The **Georgia Violent Death Reporting System (GVDRS): Case Abstraction Helper Tool** is a data abstraction and management application designed for the efficient handling of case information, including the demographic and incident details of violent death cases. This tool allows users to upload documents, view specific case information, and export data while ensuring compliance with dynamic data requirements and an adaptable table structure.

## **Key Features:**

1. **Data Display & Management**:
   * Display case information in a data grid (using MUI DataGrid).
   * The main table shows fields like PersonId, IncidentId, First Name, Certificate Number, Key, Status, and View Results.
   * A column for uploading documents and a clickable View Results link that opens detailed information in a new tab.
2. **Data Export**:
   * Integrated export functionality within the main table, allowing users to export relevant data.
3. **Dynamic Table Structure**:
   * Adaptability to dynamic fields that change with different batch data runs.
4. **Multiple Reports Handling**:
   * Handle multiple report types (e.g., Law Enforcement, Medical Examiner, Coroner Reports) per case.
5. **Victim-Focused Data Collection**:
   * Emphasis on collecting and abstracting key data about the victim, suspect, and incident circumstances.
6. **JWT Authentication**:
   * Secure authentication and authorization with JWT tokens.

## **Application Workflow**

### **1. Login Screen**

* **Technology Used**: JWT authentication with Django REST framework.
* **Flow**:
  + User inputs credentials.
  + Upon successful login, the JWT token is generated and stored.
  + User is redirected to the main data grid page.

### **2. Main Dashboard (Table View)**

* **Technology Used**: Next.js, MUI DataGrid, Axios.
* **View**:
  + The table shows all cases with the ability to upload documents directly to each row.
  + Fields displayed include PersonId, IncidentId, First Name, Certificate Number, Key, Status, and View Results.
  + The View Results column is clickable and opens a new tab with more detailed case information.
* **Document Upload**:
  + Each row has an upload feature where users can drop documents for specific cases.
  + The upload mechanism will store documents linked to the respective case on the backend.
* **Dynamic Data**: The fields in this table are dynamically populated based on the batch runs, with the potential to change as the data evolves.

### **3. Detailed Case View (Opened in New Tab)**

* **Technology Used**: Next.js, MUI Grid, Axios.
* **View**:
  + The detailed case view contains tabs for different types of data:
    - Demographics & Injury Information
    - Summary
    - Weapon & Suspect Information
    - Circumstances
  + Displays victim-related information and any injury details.
  + User can navigate through different sections of the case details using these tabs.

### **4. Backend (Django REST Framework)**

* **Technologies Used**: Django, Django REST Framework, PostgreSQL, JWT authentication.
* **API Endpoints**:
  + **Login**: Secure JWT-based login to authenticate users.
  + **Case Data API**: Fetches all case information to populate the MUI DataGrid on the frontend.
  + **File Upload API**: Receives uploaded documents and stores them in association with the case.
  + **Detailed Case View API**: Fetches detailed case data to display when the View Results link is clicked.

### **5. Export Functionality**

* **Technology Used**: Axios (for API requests).
* **Flow**:
  + Export functionality integrated into the table view.
  + Users can export individual cases as CSV or other formats from the existing page (no need for a separate export page).

## **Technologies and Libraries**

### **Frontend:**

* **Next.js**: For creating dynamic pages and components.
* **MUI DataGrid**: For rendering a dynamic, responsive table with document upload functionality and clickable rows for detailed views.
* **Yup**: For form validation in conjunction with document upload.
* **Axios**: To handle HTTP requests for fetching and uploading case data and documents.
* **TypeScript**: For type safety and maintainability in the frontend codebase.
* **ESLint**: Ensuring code quality and following best practices.

### **Backend:**

* **Django**: Main backend framework handling the application logic.
* **Django REST Framework**: For creating the REST API endpoints.
* **JWT**: Handling secure authentication and user session management.
* **PostgreSQL**: Database used for storing case information and managing records.

### **Other Important Features:**

* **Menu/Navigation**: A collapsible sidebar or top menu for navigation between different pages of the application.
* **Dynamic Table Adaptation**: Handles changes in the batch data fields dynamically and adjusts the table columns accordingly.
* **Multiple Report Handling**: Ensures the correct association of different reports (e.g., law enforcement, medical examiner) with the right case and victim.

## **Future Enhancements**

### **1. Handling of Complex Cases:**

* In cases where multiple individuals are involved in a single incident, the system will treat each individual as a separate record to ensure accurate data representation.

### **2. Multiple Source Data Extraction:**

* Data extraction from multiple sources such as vital records, law enforcement, medical examiners, and coroners will be streamlined to minimize inconsistencies and ensure accurate case details.

### **3. Batch Run Data Review:**

* Continuous review of batch run data to understand dynamic field changes and adapt the table structure accordingly.

## **Application Architecture**

### **Frontend:**

* **Next.js**:
  + Pages for Login, Main Dashboard, and Detailed Case View.
  + Dynamic routing for displaying detailed case information.
* **MUI DataGrid**:
  + A dynamic table that can handle large datasets, file uploads, and provides clickable links for viewing more detailed information.

### **Backend:**

* **Django & REST Framework**:
  + API endpoints to handle case data fetching, document uploads, and detailed case views.
  + JWT-based authentication for secure access.

This document outlines the key components, technologies, and workflows needed to build the **GVDRS: Case Abstraction Helper Tool** with modern industry standards in mind. Every aspect, from data display to export functionality and complex case management, is designed for efficiency and flexibility to accommodate dynamic field changes and multiple report types.

From analyzing both images you uploaded, here's what I understood:

### **Image 1: Main Table View**

* **Purpose:** The main interface displays a table related to the "Georgia Violent Death Reporting System: Case Abstraction Helper Tool."
* **Components:**
  + **Table Layout:** Each row in the table displays essential information about individual cases:
    - **Upload Document(s):** Provides a mechanism for document upload, allowing users to upload files for specific cases.
    - **PersonId, IncidentId, First Name:** Basic identification fields for each case.
    - **Certificate Number and Key:** Further identifiers for the case, possibly a government certificate number or unique internal case key.
    - **Status:** Indicates the processing status of the case (e.g., "In Progress").
    - **View Results:** A clickable link to view detailed case information in a separate window or tab.
* **Dynamic Data:** The table is flexible, meaning fields can change depending on the data that is processed, possibly through batch runs or dynamic reports.

**Purpose of Image 1:**

* The main dashboard serves as an overview of cases, where users can upload documents or click to see detailed views of each case. There’s a structured grid format with interactive elements such as document upload and clickable links for viewing more details.

### **Image 2: Detailed Case View**

* **Tabs for Detailed Information:**
  + **Demographics & Injury Information, Summary, Weapon & Suspect Information, Circumstances:** These tabs provide a way to break down the case into various segments, such as victim demographics, incident details, and suspect information.
  + **Displayed Fields:** This image showcases victim-related data such as:
    - **Demographics:** Displays first name, last name, age, race, ethnicity, and residence.
    - **Injury Information:** Specifies the manner of death, injury address, city, and county (though these fields are not fully filled in the example).
* **User Flow:** Clicking the "View Results" from the table would open this detailed view in a new tab, allowing users to explore specific case information through categorized tabs.

### **Overall Interface Understanding:**

* **Main Dashboard (Image 1):**
  + Users interact with a grid showing case overviews, where they can upload documents for individual cases, view case status, and open detailed reports through clickable links.
* **Detailed Case View (Image 2):**
  + Upon clicking "View Results," users are directed to a detailed tabbed interface where they can navigate through different aspects of the case, such as demographic information and incident details.

The UI needs to handle large datasets dynamically with document uploads, filtering, and clickable links for users to explore more details about a particular case in a well-organized, categorized layout.