The COMP0004 Object-Oriented Programming coursework is designed to create a website for demonstrating and manipulating provided patient data. It fulfills requirements 1-9 as specified in the question sheet. The website is equipped with functionalities to add, delete, search, and edit patient details. Notably, the functionalities for searching, deleting, and editing are consolidated into a single webpage, with adding presented in another one.

For the search/delete/edit page, initially, users are presented with 20 input boxes corresponding to each column in the DataFrame. Users can fill in any number of these boxes before the data is sent to the "model" for searching. If the entered keywords are found within the current row's fields, that row is included in the search results. On the search results page, each patient's profile link is accompanied by a "delete" button. Clicking this button prompts a confirmation window for the deletion. Accessing a patient's profile link displays a page with detailed information, where each field is editable with a “Save” button placed at the bottom of the profile page.

After making modifications, users can save the data in either .csv or .json format by clicking the corresponding button on the index page.

Additionally, the website offers features to view deceased patients and those with a driver's license.

The model package is the foundation of the application's core logic, featuring classes such as **Column** and **DataFrame** designed to organize data in a tabular format. A **DataFrame** instance encapsulates an **ArrayList** of **Column** instances, each of the same size, to represent a table. Conversely, a **Column** instance comprises a private **name** attribute and a private **ArrayList** named **row** to store data values.

To facilitate data encapsulation, which in turn enhances data integrity and security, a variety of methods are provided. These methods include **getRowValue(int row)**, **getValue(String columnName, int row)**, and **deleteRow(int row)**, allowing controlled access and manipulation of the data while keeping the underlying fields private. Additionally, all fields in these two classes are declared as **final**, ensuring their immutability post-initial assignment. This characteristic of immutability contributes to creating safer and more predictable code by eliminating the possibility of unexpected alterations to variable values.

Furthermore, the **DataLoader** and **JSONWriter** classes are implemented to enable data reading and writing functionalities, respectively. The **JSONWriter** class is designed to serialize and write data from a **DataFrame** object to a specified JSON file. It contains a **DataFrame** object and a **String** representing the name of the output file. The **JSONWriter** is equipped with a single method, **write()**, tasked with writing the **DataFrame** content to the designated JSON file. Similarly, the **DataLoader** class is associated with a **String** that represents the path to the input file. It features a singular method, **read()**, which reads the content from the specified file into a new **DataFrame** object, subsequently returning it.

The **Model** class stands as the cornerstone of the "Model" component within the Model-View-Controller (MVC) architectural pattern. It is equipped with a singular attribute of the **DataFrame** type, encapsulating the data upon which the servlets operate. This design

ensures a clear separation of concerns, adhering to the MVC paradigm by enabling servlets to interact with the application's data layer exclusively through this class.

The **Model** class provides a suite of methods that servlets can utilize to perform data operations, such as **getPatientDetails(String patientID)**, **getPatientName(String patientID)**, and **getPatientNames(ArrayList<String> patientIDs)**. These methods are designed to facilitate access to specific patient details, names, or a list of names based on provided patient IDs, streamlining the process of data retrieval for the servlets. By centralizing data access and manipulation within the **Model** class, the design effectively decouples the application's business logic from its web interface, allowing servlets to focus on handling HTTP request and response cycles, including GET and POST requests, and delegating data-related operations to the **Model**.

The JSP and HTML files play a crucial role in presenting the webpages of the application. Each JSP file is associated with a specific servlet, forming a direct link between the server-side logic and the client-side presentation. Upon completing its interactions with the Model class, a servlet prepares the data needed by the front-end by setting all relevant attributes. It then delegates the rendering task to the corresponding JSP file.

Within these JSP files, a blend of Java and HTML code is employed to dynamically generate the webpage content, displaying the processed data in a structured and aesthetically pleasing manner. This approach enables the seamless integration of server-side data with the static HTML structure, allowing for the dynamic update of content based on the user's interactions and the application's state. By leveraging this combination, the application effectively bridges the gap between the back-end logic and the front-end presentation, ensuring that the data is displayed accurately and responsively to the end users.

The project abstracts complex operations into discrete classes like **DataFrame** and **Model**, allowing for detail hiding and interface simplification. For example, data operations are abstracted behind the model package, while client interactions are managed through servlets, presenting a clean interface to both data handling and web interaction functionalities.

Classes within the project are highly cohesive, focusing on specific responsibilities. For example, the model package classes are solely concerned with data representation and operations, whereas the servlets package classes focus on processing web requests. This high level of cohesion facilitates understanding, testing, and maintenance of the code.

Overall, the coursework project has been designed with a clear structure that adheres to Object-Oriented (OO) principles, demonstrating thoughtful organization and application of key OO concepts.