GRASP Patterns in our web app

- Link to GitHub project repository: https://github.com/zayatMark/3220-group-project

- *Controller*: this pattern handles user interactions and system events by deciding which components to display and what actions to take in an application.
- Example: the **App class** in **App.js**, since within its **render() method** (lines 21-39) it manages system events and user interactions and centralizes the control logic in a single entity and renders whatever the user wants to display. Basically, this helps keep the app organized and easy to change or grow, making updates and testing simpler.

- *Creator*: this pattern involves a class creating other objects or instances of that class when it has the information necessary to do so. Promotes encapsulation.
- Example: the **MultiDataView class** in **multidata.js**, since it creates instances of the Classification class (see line 28). Essentially, this method has the required information to create Classification objects, is the class that stores the object and uses them for data processing tasks (for instance, when fetching or filtering data) upon the user's requests.

- *Information Expert*: this pattern assigns responsibilities to the class that has the specific knowledge needed to perform a given task.
- Example: While there are many examples of this GRASP pattern in our **Classification class**, the **filterData() method** in that class in **filecomponents.js** is one of many instances (lines 106-164), since it is responsible for filtering data based on criteria provided by the values parameter. It stores the information on the data that needs to be filtered, making it an information expert on this data. Putting all data filtering and handling in the Classification class makes the app less complicated and easier to manage.

- *Low Coupling*: this pattern aims to minimize dependencies between different parts of an application, making it easier to change one part without affecting others.
- Methods such as **getListOfFiles()** (lines 87-99) in the **Classification class** demonstrate an example of the Low Coupling approach since it reminds us that the MultiDataView class and other UI classes don't actually need to interact with the logic in Classification, which contains the getListOfFiles() method, to display information. In other words, the responsibility of retrieving and filtering the data is

neatly contained in an independent class that is accessible by others when required, rather than each class needing to know how this implementation works.

- *High Cohesion*: this pattern ensures that a class is focused on a single purpose or closely related activities, making the class more understandable and manageable.
- The **DisplayCSVData class** in **DisplayCSVData.js** (lines 5-84) demonstrates High Cohesion since it's dedicated to a main task: parsing, loading, and displaying the data from a .csv file for a given item, making the class very focused on showing the csv on screen. This has many benefits, like more organized and modular components, more readable code, and easier modification.