

# 1 Purpose

The purpose of this document is to define how the binding and filtering algorithm works within the Region 15 Scholarship Application. As such, it must define how the student and provider schema works and is editable by the application administrators.

## 2 Student and Provider Schema

Administrators of the program shall be able to define a specific structure for the data that should be collected from the students and providers. This requires the ability to edit forms based on a series of options.

### 2.1 Data Types

The application shall have a series of data types, so far limited to:

- Text
- Numbers
- Lists (multiple text or number values)
- Mapped values (larger structures comprised of multiple separate text, number, or list values)

### 2.2 Input Types

The application has a series of components that have been or will be built to support this data collection:

- `OutlinedTextField`: supports text and numbers
- `Dropdown`: supports strictly defined text
- `CheckboxList`: supports strictly defined text lists
- `RadioList`: supports strictly defined text
- `MultiEntry`: supports mapped values

Student information is defined within a specific form schema. The required values for each input type to work is well-defined, and shall be editable by the scholarship administrators. An interface shall be created that allows for editing with a simple preview.

## 3 Student and Provider Binding

For the purposes of this document, binding is defined as the ability to relate data members defined in the student and provider data structures so that they may be procedurally or programmatically filtered through a database. For example, binding the `first_name` member from the student with the `scholarship_name` member from the provider means that the application administrators shall be able to perform comparisons from the `first_name` to the `scholarship_name`, or vice versa.

### 3.1 Data Types in comparisons

Values that do not have the same type shall not be bindable. For example, a member that contains a text value can *only* be bound with another member that also contains text. The same applies for numbers and lists.

Mapped values are slightly different, in that they contain a series of extra values. Since mapped values contain other data types, binding must happen on a more precise scale. The administrators shall be able to choose individual members from within the map and link them to other members of the same type. For example, a data map `example_map` contains a member `text_member`. Binding `text_member` is only possible with another member that contains text.

## 4 Methodology

### 4.1 Constraints

Administrators will create a series of comparisons dependent upon the types of the members that are being bound. For example, number values will present inequality options. Text values are a little more complex, in that they may have a specific set of options that are available (users are shown a `RadioList` or `Dropdown`). Depending on what these values are defined as, the available options for comparison need to match.

### 4.2 Appearance

Administrators will be able to select a specific schema - provider or student - and the members will be displayed in a list-like format. Depending on the type of each member shown, the available options for comparisons will follow the rules above. The administrators can choose and create multiple of these conditions, which will be displayed to the providers while they are choosing requirements.

Comparisons will have a member that contains a “display text.” This text is displayed to the providers and acts as an abstraction for the actual comparison. For example, a GPA comparison may consist of the `Student.gpa` member, a `>` condition, and a conditional value of `3.2`. The display text for this comparison may be “GPA > 3.2”, or anything that the administrator may choose.

After being created, comparisons will be sortable within the list that the providers will see. The list may have multiple categories and multiple conditions may be selected by the providers.

Members with associated comparisons shall have a boolean value that indicates whether multiple conditions are selectable. For example, perhaps a provider wants to select multiple sports (athletics has several comparisons, one for each sport). The athletics member will be able to determine that multiple values are selectable, allowing the providers to select multiple types of sports.

**NOTE: The issue with this is that even though multiple options are selected, we do not currently have a method to determine if the options are joined with an AND or OR condition. This means that selecting multiple sports does not distinguish between a student playing football AND soccer, or a student playing football OR soccer.**