K-Lab Coalesce

Technical Report

**Rishi Prasad Pisipati**

Carnegie Mellon University  
METALS 2018-19

# Summary of Product

Over the course of the past 8 months, we have collaborated with K12 Inc, one of America’s largest online schools to develop a tool that would scaffold Project Based Learning. Based on a mutual agreement after the spring semester, we decided to focus our attention towards “Personalized Collaboration”. Within that domain, off the components that were designed as clickable prototypes, we were successfully able to implement two major components up to the point of a functional prototype.

* Teams API data extraction
* Grouping tool **(Main component)**

This report will detail the finer details associated with each component. Each component will be accompanied with a high level diagram depicting the relationship between all of the components. The bulk of this report will be related to the grouping tool.

# Table of Contents

[Summary of Product 2](#_Toc15875474)

[Table of Contents 3](#_Toc15875475)

[Framework and Setup Instructions 5](#_Toc15875476)

[Teams API Data Extraction 6](#_Toc15875477)

[Grouping Tool 7](#_Toc15875478)

[Documentation 9](#_Toc15875479)

[HTML 9](#_Toc15875480)

[button\_row 9](#_Toc15875481)

[parent-container 10](#_Toc15875482)

[group-container 12](#_Toc15875483)

[TypeScript 13](#_Toc15875484)

[student.ts 13](#_Toc15875485)

[grouping.ts 13](#_Toc15875486)

[Complex CSS Components 20](#_Toc15875487)

[leaf 20](#_Toc15875488)

[line 23](#_Toc15875489)

[Contact 25](#_Toc15875490)

# Framework and Setup Instructions

Coalesce built with ❤️ using Angular8, SortableJS, PrimeNG, Yarn and Bootstrap

**Setup**

Fork the repository by executing the following command on your git terminal.

Navigate to the folder in which you have forked this repository. Run the following command  
  
npm install all

to install all dependencies required for the project.   
  
**Production server**

Run the following command on your terminal  
  
ng serve  
  
Navigate to htttp://localhost:4200 to view a remote version of the website. Any changes you make to the code will automatically be deployed to the remote server.  
  
**Deployment**

Run the following command to build the final app.  
  
ng build  
  
Host the application on a cloud service of your choice.

# Teams API Data Extraction

The detailed documentation for this component can be accessed [here](https://docs.microsoft.com/en-us/graph/api/channel-list-messages?view=graph-rest-beta&tabs=http).

In order to access this component, an account needs to be provided with the appropriate permissions in order to access this information. An authentication token is required to authorize the request.

**Permissions:**  
  
Group.Read.All,Group.ReadWrite.All  
  
  
**Sample request**  
  
GET /teams/{id}/channels/{id}/messages  
  
**Sample code for Javascript**

const options = {

authProvider,

};

const client = Client.init(options);

let res = await client.api('/teams/303d2c1c-f1c5-40ce-b68e-544343d7f42b/channels/19:fec4b0f2825d4c8c82abc09027a64184@thread.skype/messages')

.version('beta')

.get();

**Optional Parameters**

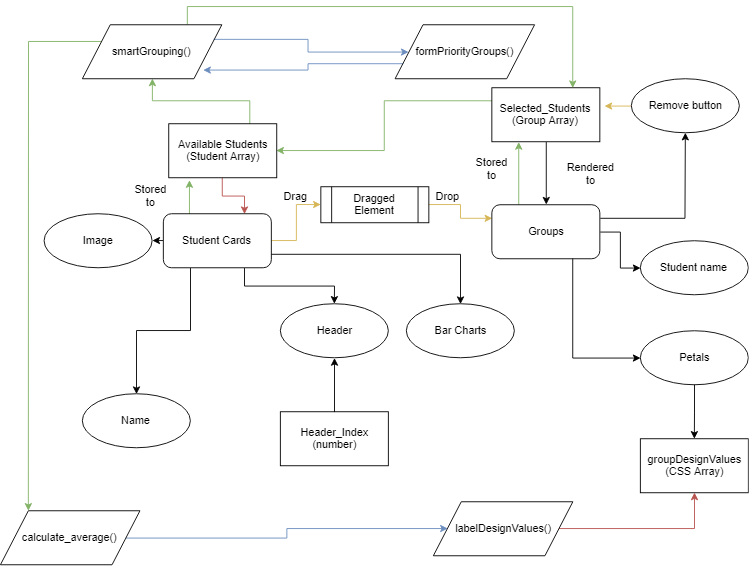
As of now, Teams does not support OData query parameters for this component. You can however, use the $top query parameter to retrieve a finite number of queries. In all likelihood, you may have to clip the parameters in order to facilitate data extraction.

**Progress**As K12 could not provide us with an account with access to this component, the work on this component is currently archived.

# Grouping Tool

The grouping tool has the following top-level architecture :  
Note : Red lines depict storing into variables, blue lines depict interactions within functions, black lines indicate components rendered to the UI, orange lines depict interactions with the data/components.

Parallelograms represent core functions, rounded rectangles represent core front end components, and circles represent attributes within core front end components and sharp rectangles represent where the data is stored.



**Key Components**

# Documentation

Further explanation of all elements associated with the grouping tool is available in the source code. This documentation is intended to give a brief overview associated with all components.

## HTML

**hover\_background\_grouping**Div responsible for rendering the popup when “Configure grouping” is pressed. It is rendered with an \*ngIf that is set to true when the button is clicked. An overlay is added with the div centered around the available students. Has a list of sortableJS items associated with the given div that can be reordered according to the user’s preference.

### button\_row

Div responsible for rendering the buttons in the HTML. Has a total of 4 buttons -   
1. Sort by  
2. Configure Grouping  
3. Smart Grouping  
4. Reset Grouping

#### **dropbtn**

Button responsible for sorting the available student roster. Has a hidden drop down list that is revealed when hovered on. Can be configured to sort students by Collaboration, Communication, Problem Solving, Group Discussion, Project Management. Each dropdown is a link that invokes the sortAttributes function, each with it’s own integer parameter associated with the strings.

#### **configure-grouping**

Button responsible for generating the hover\_background\_grouping div. Invokes a function that changes the state of a variable used in the ngIf parameter in hover\_background\_grouping. Generates the popup allowing the user to configure the drag and drop functionality.

#### **smart-grouping**

Button responsible for generating smart groups. Invokes a function that is responsible for the higher level architecture of the smart grouping function.

#### **reset-groups**

Button responsible for resetting the groups teacher or the smart grouping algorithm has formed. Invokes a function that is responsible for resetting groups to

### parent-container

Contains all of the components associated with the student and group cards. Occupies the remainder of the height and full width of the screen.

#### drag-column

Contains all the elements associated with the student grouping card. Occupies a width of 70% from the parent container. This is initialized with a grid view and has 3 equidistant columns on which the cards will render.

##### **ui-helper-clearfix.ng-star-inserted**

Generates the components associated with creating the student cards. Cards are generated using a ngFor directive based off the student data. Each card occupies 85% of the grid width to ensure that cards do not stick together. This div has two additional subcategories.

* student-profile-info
* student-performance

###### **student-description**

A div containing a student’s name and student’s image source. Data for both components is accessed using typescript.

###### **student-performance**

A div containing all the horizontal bar charts associated with student performance along with a shorthand description of each line.   
Col – Collaboration  
Com – Communication

P.S – Problem Solving  
P.M – Project Management

G.D – Group Discussion

Each of the prefixes is followed by a horizontal bar graph generated using CSS. The width of the graphs is determined dynamically at runtime by the label\_to\_percentage function which passes the student’s performance as a parameter.

### group-container

Contains the group elements. Set to a width of 30% on the right. It contains the edit students button along with the save and confirm button.

#### **drag-column**

Contains the grouping elements. Each card has two containers.  
1. Ng-template containing student roster

2. group-avg-score

##### remove-button

Contains the button used for removing the student from the group and adding it back to the student roster.

##### **group-avg-score**

Contains the petals used for rendering the group average. Each petal is denoted by the leaf class with leaf class (1-n).

###### **leaf (petal)**

Each petal (leaf) is positioned absolutely within the scope of the div. The leaf structure is created using css and is rotated to the right order. This will be elaborated further in the CSS documentation.

## TypeScript

### student.ts

The student class is declared in student.ts  
Each instance of the student class is initialized with the following members

* Image-source : string
* Name : string
* Gender : string
* Collaboration : string
* Communication : string
* Problem : string
* Group\_discussion : string
* Project\_management : string
* Student\_header : Array of strings, initialized with the values passed to the five categories   
  (Collaboration, Communication, Problem Solving, Group Discussion, Project Management)

This data structure allows for easy access and manipulation of student elements.

### grouping.ts

Grouping.ts is a large file with all the interactions for the grouping tab contained into one typescript file. The class exported is called GroupingComponent and has the following components.

#### **Variables**

There are several variables initialized in this file

availableStudents : Array of student class. Used in the individual card container.

selectedStudents: An array of array of students. Used in the grouping container.

draggedStudent: Student class variable, used to denote dragged student.

numberOfGroups : number, denotes the number of groups

group\_Placeholder\_Container : An array of array of grouping containers

header\_index : number, denotes the current header for student card

studentsPerGroup : number representing the number of students in group

groupDesignValues : An array of array of group design values

toggleInputValue : boolean, toggles the input box for studentsPerGroup

toggleOverlayGrouping : Boolean, toggles the input

#### **Functions**

##### **ngOnInit()**

Initializes all the variables with their required values. Contains a variable called dataSource that is initialized with the student roster. Pushes the student roster to availableStudents and the default number of groups to the group\_Placeholder\_container. We also sort the student roster and initialize those rosters into groupingStrings. This is also used in the popup for ordering the student roster.

##### **save\_function()**

Creates a popup that says that the grouping is saved. Currently does not save the value to any file as it has no added value, since the data needs to be stored reliably on a server.

##### **confirm\_function()** return : none

Creates a popup that says that the grouping is confirmed. Currently does not save the value to any file as it has no added value, since the data needs to be stored reliably on a server. Should ideally set the student roster and store it onto K12’s LMS.

##### **calculate\_average(index : number)** return : none

Calculates the average value assigned to each group. The index parameter is used to reference the array in selectedStudents.   
The average is calculated for each of the 5 categories by assigning a numeric score to each value from the collaboration score document. It is then averaged out by the number of students currently in the given roster. Calls labelDesignValues after the average for each category is calculated, and initializes the same into groupDesignValues

##### **labelDesignValues(value : number)** return : strings Returns a string based on the given value. 75-100 : return “exemplary” 50-75 : return “accomplished” 0-50 : return “developing”

##### **dragStart($event, Student)**

Sets draggedStudent to the parameter of Student. Initializes the dragged student after the student card is dragged. Uses primeNG’s functionality

##### **drop($event, index)**

return type : None  
Initializes the drop functionality using primeNG’s library. Sets the student to the appropriate group using the index as a parameter. Saves the value to this.selectedStudents[index]

##### **dragEnd($event)**

return type : None

Sets draggedStudent to null.

##### **findIndex(student: Student)**

return type : number

Returns the index of the student after the student is found in the individual card roster.

##### **findGroupIndex(student : Student, index : number)**

return type : number

Returns the index of the student within a certain group. Uses the index parameter to locate the group in selectedStudents.

##### **removeGroupValue(student : Student, index : number)**

return type : None

Removes the student from the group using the index as a parameter to locate the group in selectedStudents. Reassigns them to availableStudents. Calls the calculate\_average function to reassign the petal values.

##### **reset\_groups()**

returns None  
Assigns all the students currently in groups stored in selectedStudents back to availableStudents.

##### **increment\_groups()**

return None  
Increments the number of groups by 1. Initializes a blank group at the end of the container by assigning a blank value to the end of selectedStudents and groupDesignValues.

##### **label\_to\_percentage(value : string)**

returns String

Returns a string based on the value given to label\_to\_percentage. This value is then used as an ngStyle variable for the bar charts.   
“Exemplary” returns “100%”

“Accomplished” returns “75%”

“Developing” returns “50%”

##### **label\_to\_value (value: string)**

returns number

Returns a string based on the value given to label\_to\_percentage. This value is then used as an ngStyle variable for the bar charts.

“Exemplary” returns 100

“Accomplished” returns 75

“Developing” returns 50

This is used by the calculate\_averge function to calculate the average of each component.

##### **sortAttributes(value : number)**

returns None  
Sorts the availableStudents variable based on the value passed. This is used by the dropdown button used to sort groups.  
1 : Sorts by Collaboration  
2: Sorts by Communication

3: Sorts by Group Discussion

4: Sorts by Project Management

5: Sorts by Group Discussion

##### smart\_grouping()

returns None  
The master function for forming groups. Sets all the assigned students, if any to availableStudents. Retrieves the priority list from groupingStrings. Passes the student roster, studentsPerGroup and priority list to formPriorityGroups. Assigns the returned value from formPriorityGroups to selectedStudents.  
Uses the calculate\_average function to assign values to each of the petals.

##### formPriorityGroups(studentData : Student[], studentsperGroup : number, priority\_list : Array)

returns Array of Student[]

Uses the priority list to ascertain how Smart Grouping will work. Returns the grouped list of Student arrays.  
**Algorithm**

* Arrange student roster into 5 copies ordered based on the priorities.
* Use the first priority category to assign the pioneer in each group. Filter those students out from the other 4 copies of student rosters.
* Choose the highest priority element from the remainder of the rosters starting with priority 2-5 and assigning them to groups 1->n. Cycle through the categories using (i%4) and cycle through the groups using (i%no\_of\_groups) in even rounds.
* Once a round is over, reverse the direction of assignment by assigning priority 2-5 over groups n->1, this prevents the last group from always getting the weakest students. Achieve this through N – (i%no\_of\_groups) for odd rounds.
* This continues until students/per group is satisfied. In the event students are left over in the roster (always n-1 students), we calculate the weakest teams and sort in ascending order. The accomplished students are then assigned sequentially into the weakest teams until the group rosters are formed.

For a walkthrough of what this categorization would look like for 3 categories, watch this video.

# Complex CSS Components

In this section, I will cover some of the complex CSS components used to render the grouping tool.

## leaf

.leaf {

position: absolute;

margin-left : 18%;

width: 40px;

height: 40px;

background:white;

border-radius: 40px 0;

transform-origin: left bottom;

}

Leafs need to be positioned using the absolute parameter, else they render in very random places over the page. The background of the leaf class needs to be set to white so as to initialize each individual petal class with the appropriate colors. Border-radius gives the shape of the petals by shaving the petals to the same width as .

.leaf-1 {

transform: rotate(-45deg) translate(4px, -4px);

/\* background: #A9B9FF; \*/

border : 2px solid #A9B9FF;

}

.leaf-2 {

transform: rotate(27deg) translate(4px, -4px);

/\* background: #F47645; \*/

border : 2px solid #F47645;

}

.leaf-3 {

transform: rotate(99deg) translate(4px, -4px);

/\* background: #F7AA01; \*/

border : 2px solid #F7AA01;

}

.leaf-4 {

transform: rotate(171deg) translate(4px, -4px);

/\* background: #7ACFDE; \*/

border: 2px solid #7ACFDE;

}

.leaf-5 {

transform: rotate(243deg) translate(4px, -4px);

/\* background: #FFA2DD; \*/

border: 2px solid #FFA2DD;

}

Through a lot of trial and error, the appropriate angles were assigned to the petals by using the rotate property. The transform property is used to reposition the elements on the x and y axis so as to bring the petals closer.

.leaf-1.exemplary

{

background: #A9B9FF;

transform: rotate(-45deg) scale(1.0) translate(4px, -4px);

}

.leaf-2.exemplary {

transform: rotate(27deg) scale(1.0) translate(4px, -4px);

background: #F47645;

}

.leaf-3.exemplary{

transform: rotate(99deg) translate(4px, -4px);

background: #F7AA01;

}

.leaf-4.exemplary {

transform: rotate(171deg) translate(4px, -4px);

background: #7ACFDE;

border: 2px solid #7ACFDE;

}

.leaf-5.exemplary{

transform: rotate(243deg) translate(4px, -4px);

background: #FFA2DD;

border: 2px solid #FFA2DD;

}

.leaf-1.accomplished

{

background: #A9B9FF;

transform: rotate(-45deg) scale(0.75) translate(4px, -4px);

}

.leaf-2.accomplished {

transform: rotate(27deg) scale(0.75) translate(4px, -4px);

background: #F47645;

}

.leaf-3.accomplished{

transform: rotate(99deg) scale(0.75) translate(4px, -4px);

background: #F7AA01;

}

.leaf-4.accomplished {

transform: rotate(171deg) scale(0.75) translate(4px, -4px);

background: #7ACFDE;

border: 2px solid #7ACFDE;

}

.leaf-5.accomplished{

transform: rotate(243deg) scale(0.75) translate(4px, -4px);

background: #FFA2DD;

border: 2px solid #FFA2DD;

}

.leaf-1.developing

{

background: #A9B9FF;

transform: rotate(-45deg) scale(0.5) translate(4px, -4px);

}

.leaf-2.developing {

transform: rotate(27deg) scale(0.5) translate(4px, -4px);

background: #F47645;

}

.leaf-3.developing{

transform: rotate(99deg) scale(0.5) translate(4px, -4px);

background: #F7AA01;

}

.leaf-4.developing {

transform: rotate(171deg) scale(0.5) translate(4px, -4px);

background: #7ACFDE;

border: 2px solid #7ACFDE;

}

.leaf-5.developing{

transform: rotate(243deg) scale(0.5) translate(4px, -4px);

background: #FFA2DD;

border: 2px solid #FFA2DD;

}

Finally, the classes for each of the 5 leaf classes need to be permuted with the set of class values of “Exemplary”, “Accomplished” and “Developing”. This is the step where you fill in the colors into the petals. The scale property is used to resize the petals based on the class. Classes are dynamically assigned to each petal during runtime using the ngStyle property.

## line

.line

{

min-height: 15px;

height : 5%;

margin-left: 20%;

box-sizing: border-box;

background: #000;

margin-top : -20%;

}

The line class is initialized with a border-box and each box is assigned a color of black. Each line is then initialized with a height of 15% or a min-height of 5%.

.line-1

{

background: #A9B9FF;

}

.line-2

{

background: #F47645;

}

.line-3

{

background: #F7AA01;

}

.line-4

{

background: #7ACFDE;

}

.line-5

{

background: #FFA2DD;

margin-bottom: 15px;

}

Each of the lines is then initialized with it’s own background color to represent different colors.

# Contact

If you’re interested in any further elaboration of components in this tool, please feel free to reach out to me at [rishiprsd@gmail.com](mailto:rishiprsd@gmail.com)

I am actively looking for jobs in Web and Back-end development.

To see more of my work, visit : <http://www.rpisipat.com>