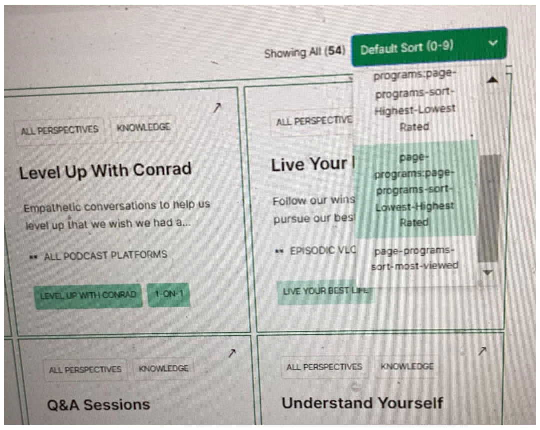
This allows sorting options to be seen in the dropdown and sets a value which will be pulled when running useProgramsTable2.tsx.
index.tsx lines 446-474

```
        <Th>
          <StyledSelect
            className="react-select-container"
            classNamePrefix="react-select"
            options={[
              {
                label: t("page-programs:page-programs-sort-by"),
                options: [
                  // Add an option for default sorting
                  { label: t("page-programs:page-programs-sort-default"), value: 'default' },
                  // Add an option for alphabetical sorting
                  { label: t("page-programs:page-programs-sort-alphabetical"), value: 'alphabetical' },
                  // Highest to Lowest Rated sorting option
                  { label: t("page-programs:page-programs-sort-Highest-Lowest Rated"), value:
'highestRated' },
                  // Lowest to Highest Rated sorting option
                  { label: t("page-programs:page-programs-sort-Lowest-Highest Rated"), value:
'lowestRated' },
                  // PopularMost Viewed sorting option
                  { label: t("page-programs:page-programs-sort-most-viewed"), value: 'mostViewed' },
                ],
              },
            ]}
            placeholder={t("page-programs:page-programs-sort-by")}
            onChange={(selectedOption) => {
              updateSortOrder(selectedOption);
            }}
            defaultValue={{ label: t("page-programs:page-programs-sort-default"), value: 'default' }}
// Set the default option
            isSearchable={false}
          />
        </Th>
```



Empathetic conversations to help us level up that we wish we had a...

Follow our wins pursue our best

The useProgramsTable2.tsx file will receive various changes to incorporate the new sorting options, here is the code for that. Currently only includes lowest to highest rated and highest to lowest rated sorting. This feature does not work as we can't pull an average rating so this can be more used as the pseudo code for now but should fully compile still. Assuming there is content that has not been rated yet, it will go to default sort even when sorting by average ratings. The next iteration will include Popular/Most Visited by pulling Youtube's API.

useProgramsTable2.tsx

```tsx
import { useEffect, useMemo, useState } from "react"
import { Icon } from "@chakra-ui/react"

import { FrameworkTableProps } from "@/components/Programs/ProgramsTable"

import ProgramsDropdownItems from "./ProgramsDropdownItems"

export interface DropdownOption {
  label: string
  value: string
  description: string
  filterKey: string
  category: string
  icon: typeof Icon
}

export type ColumnClassName = "firstCol" | "secondCol" | "thirdCol"

type UseFrameworkTableProps = Pick<FrameworkTableProps, "filters" | "frameworkData"> & {
  t: (x: string) => string
  selectedTags: string[]
}

export const useFrameworkTable = ({
  filters,
  selectedTags,
  t,
  frameworkData,
}: UseFrameworkTableProps) => {
  // Add a state to manage the sort order
  const [sortOrder, setSortOrder] = useState('default');

  const { featureDropdownItems, perspectiveDropdownItems } = ProgramsDropdownItems({ t });

  const filteredFrameworks = useMemo(() => {
    // Start with the full frameworkData array and apply the filter logic
    let frameworksToProcess = frameworkData.filter((framework) => {
      // console.log('Starting filter operation');
      // console.log('Current filters:', filters);
      // console.log('Selected tags:', selectedTags);
      // console.log('Evaluating framework:', framework.title);

      // Group feature filters by category
      const filtersByCategory = featureDropdownItems.reduce((acc, item) => {
        const { category, filterKey } = item;
        if (!acc[category]) {
          acc[category] = [];
        }
        if (filters[filterKey]) {
          acc[category].push(filterKey);
        }
        return acc;
      }, {});

      // Check for feature filter match within each category (OR relationship)
      const featureMatchWithinCategories = Object.keys(filtersByCategory).every(category => {
        return filtersByCategory[category].length === 0 || filtersByCategory[category].some(filterKey
=> {
          return framework[category] === filterKey;
        });
      });

      // Check if any tag is selected
      const isAnyTagSelected = selectedTags?.length > 0;
      // console.log('Any tag selected:', isAnyTagSelected);

      // Check for tag filter match
      const tagsMatch = isAnyTagSelected ? selectedTags.every(tag => {
        return framework.tags?.includes(tag);
      }) : true;

      // Framework must match feature filters across all categories (AND relationship) and tag filters
      return featureMatchWithinCategories && tagsMatch;
    });

    // Sort the frameworks based on sortOrder, handling potentially undefined frameworkLevel
    // and sorting programType by 'knowledge', 'action', and then 'community'

    if (sortOrder=== 'alphabetical')
        {
            frameworksToProcess.sort((a, b) => a.title.localeCompare(b.title));
        }
    else if (sortOrder==='default')
        {
        frameworksToProcess.sort((a, b) => {
          const levelA = a.frameworkLevel || '';
          const levelB = b.frameworkLevel || '';
          const levelComparison = levelA.localeCompare(levelB);
          if (levelComparison!== 0) return levelComparison;

          const typeOrder = { 'knowledge': 1, 'action': 2, 'community': 3 };
          const typeA = typeOrder[a.programType] || 4;
          const typeB = typeOrder[b.programType] || 4;
          return typeA - typeB;
        });
        }
        else if (sortOrder==='highestRated')
            {
            frameworksToProcess.sort((a, b) => {
              const ratingA = a.averageRating || 0; // assume 0 rating if not available
              const ratingB = b.averageRating || 0;
              if (ratingA === 0 && ratingB === 0)
```

```
                {
                // If both ratings are 0, sort based on default order
                    return (a.frameworkLevel || '').localeCompare(b.frameworkLevel || '');
                }
                else
                {
                    return ratingB - ratingA;
                }
            });
            }
        else if (sortOrder==='lowestRated')
            {
            frameworksToProcess.sort((a, b) => {
              const ratingA = a.averageRating || 0; // assume 0 rating if not available
              const ratingB = b.averageRating || 0;
              if (ratingA === 0 && ratingB === 0)
                {
                // If both ratings are 0, sort based on default order
                    return (a.frameworkLevel || '').localeCompare(b.frameworkLevel || '');
                }
                else
                {
                    return ratingA - ratingB;
                }
            });
            }

      return frameworksToProcess;
    }, [frameworkData, selectedTags, filters, sortOrder]);

  // Add a function to update the sort order
const updateSortOrder = (selectedOption: DropdownOption) => {
    setSortOrder(selectedOption.value);
  };

  return {
    filteredFrameworks,
    updateSortOrder,
  }
}
```