```
import { useCallback, useEffect, useMemo, useState } from 'react';
import { debounce } from '@/shared/utils/helpers';
import { trpc } from '@/shared/utils/trpc/trpc';
import { defaultOptions, explorerOptions } from '@/shared/constants/explorer';
import { useSearchParams } from 'next/navigation';

const promptLimit = 6;

export default function useModels() {
  const searchParams = useSearchParams();
  const categoryQuery = searchParams?.get('category');
  const searchQuery = searchParams?.get('search');

  const [promptOffset, setPromptOffset] = useState(0);
  const [prompts, setPrompts] = useState<any[]>([]);
  const [isFetchingPrompts, setIsFetchingPrompts] = useState(false);
  const [search, setSearch] = useState('');

  const modelsQuery = trpc.explorer.getModels.useQuery();
  const toolsQuery = trpc.explorer.getAllTools.useQuery();
  const swarmsQuery = trpc.explorer.getAllApprovedSwarms.useQuery();
  const promptsQuery = trpc.explorer.getAllPrompts.useQuery({
    limit: promptLimit,
    offset: promptOffset,
    search: searchQuery || search,
  });
```

```
const agentsQuery = trpc.explorer.getAllAgents.useQuery();
const pendingSwarms = trpc.explorer.getMyPendingSwarms.useQuery();

const isDataLoading =
  modelsQuery.isLoading &&
  swarmsQuery.isLoading &&
  promptsQuery.isLoading &&
  agentsQuery.isLoading;

const [options, setOptions] = useState(defaultOptions);
const [filterOption, setFilterOption] = useState<string>(
  explorerOptions[0].value,
);

useEffect(() => {
  if (searchQuery && categoryQuery) {
    setSearch(searchQuery);
    setFilterOption(categoryQuery);
  }
}, [searchQuery, categoryQuery]);

useEffect(() => {
  if (promptsQuery.data?.data) {
    if (promptOffset === 0) {
      setPrompts(promptsQuery.data?.data);
    } else {
      setPrompts((prev) => [...prev, ...promptsQuery.data?.data]);
    }
    setIsFetchingPrompts(false);
  }
}, [promptsQuery.data?.data, promptOffset]);

const loadMorePrompts = useCallback(() => {
  setPromptOffset((prevOffset) => prevOffset + promptLimit);
  setIsFetchingPrompts(true);
}, []);

const debouncedSearch = useMemo(() => debounce(setSearch, 0), []);

const handleSearchChange = useCallback(
  (value: string) => {
    setPromptOffset(0);
    debouncedSearch(value);
  },
  [debouncedSearch],
);

// TODO: Add types
const filterData = useCallback(
  (data: any, key: string) => {
    if (!data) return [];
    if (filterOption === 'all') {
      return data.filter(
        (item: any) =>
          item?.name?.toLowerCase().includes(search.toLowerCase()) ||
          item?.prompt?.toLowerCase().includes(search.toLowerCase()),
      );
    }
    if (!search || filterOption !== key) return data;
    return data.filter(
      (item: any) =>
        item?.name?.toLowerCase().includes(search.toLowerCase()) ||
        item?.prompt?.toLowerCase().includes(search.toLowerCase()),
    );
  },
  [search, filterOption],
);

const filteredModels = useMemo(
  () => filterData(modelsQuery.data?.data, 'models'),
  [modelsQuery.data, filterData],
);
const filteredSwarms = useMemo(
  () => filterData(swarmsQuery.data?.data, 'swarms'),
  [swarmsQuery.data, filterData],
);
const filteredPrompts = useMemo(
  () => filterData(prompts, 'prompts'),
  [prompts, filterData],
);
const filteredAgents = useMemo(
  () => filterData(agentsQuery.data?.data, 'agents'),
  [agentsQuery.data, filterData],
);
const filteredTools = useMemo(
  () => filterData(toolsQuery.data?.data, 'tools'),
  [toolsQuery.data, filterData],
);

const handleOptionChange = useCallback(
  (value: string) => {
    if (isDataLoading) return;

    setFilterOption(value);
  },
  [isDataLoading],
);

return {
  filteredModels,
  filteredSwarms,
  filteredPrompts,
  filteredAgents,
  filteredTools,
    pendingSwarms,
    allPrompts: promptsQuery,
    allAgents: agentsQuery,
    allTools: toolsQuery,
    isPromptLoading: promptsQuery.isLoading,
    isModelsLoading: modelsQuery.isLoading,
    isAgentsLoading: agentsQuery.isLoading,
    isSwarmsLoading: swarmsQuery.isLoading || pendingSwarms.isLoading,
    isToolsLoading: toolsQuery.isLoading,
    search,
    options,
    hasMorePrompts: prompts.length > promptOffset,
    filterOption,
    isDataLoading,
    isFetchingPrompts,
    loadMorePrompts,
    handleSearchChange,
    handleOptionChange,
  };
}
```