```
import { createSlice, createAsyncThunk } from "@reduxjs/toolkit";
import hoSoService from "services/RegulationAndDocument/HoSoService";
import { cloneDeep } from "lodash";

export const getHoSoManage = createAsyncThunk(
  "hoSoManage/getHoSoManage",
  async (data, { rejectWithValue }) => {
    try {
      const response = await hoSoService.GetManage(data);
      return response.data;
    } catch (err) {
      console.error("API call failed:", err);
      return rejectWithValue(err.message || "Error");
    }
  }
);

export const getHoSoManageById = createAsyncThunk(
  "hoSoManage/getHoSoManageById",
  async (data, { rejectWithValue }) => {
    const { onSuccess, id } = data;
    try {
      const response = await hoSoService.GetManageById(id);
      if (onSuccess) onSuccess(response);
      return response.data;
    } catch (error) {
      console.error("API call failed:", error);
      return rejectWithValue(error.message || "Error");
    }
  }
);

export const updateHoSoManage = createAsyncThunk(
  "hoSoManage/updateHoSoManage",
  async (data, { rejectWithValue, dispatch }) => {
    try {
      const { onSuccess } = data;
      const payload = cloneDeep(data);
      delete payload.onSuccess;
      const response = await hoSoService.UpdateManage(data);
      if (onSuccess) onSuccess(response);

      dispatch(getHoSoManageById({ id: response.data }));

      return response.data;
    } catch (error) {
      console.error("API call failed:", error);
      return rejectWithValue(error.message || "Error");
    }
  }
);

export const createHoSoManage = createAsyncThunk(
  "hoSoManage/createHoSoManage",
  async (data, { rejectWithValue, dispatch }) => {
    try {
      const { onSuccess } = data;
      const payload = cloneDeep(data);
      delete payload.onSuccess;
      const response = await hoSoService.CreateManage(data);
      if (onSuccess) onSuccess(response.data);

      dispatch(getHoSoManageById({ id: response.data?.id }));
      return response.data;
    } catch (error) {
      console.error("API call failed:", error);
      return rejectWithValue(error.description || "Error");
    }
  }
);
```

```
export const deleteHoSoManage = createAsyncThunk(
  "hoSoManage/deleteHoSoManage",
  async (data, { rejectWithValue }) => {
    const { onSuccess, id } = data;
    try {
      const response = await hoSoService.DeleteManage(id);
      if (onSuccess) onSuccess(response);
      return id;
    } catch (error) {
      console.error("API call failed:", error);
      return rejectWithValue(error.message || "Error");
    }
  }
);

const initialState = {
  hoSoManageList: {
    data: [],
    loading: false,
  },
  hoSoManage: {
    data: null,
    loading: false,
  },
  initFormHS: {
    name: null,
    number: null,
    createdAt: null,
    danhMucId: null,
    description: null,
  },
};

const hoSoManageSlice = createSlice({
  name: "hoSoManage",
  initialState,
  reducers: {
    updateFormHS: (state, action) => {
      state.initFormHS = {
        ...state.initFormHS,
        ...action.payload,
      };
    },
    resetFormHS: (state) => {
      state.initFormHS = initialState.initFormHS;
    },
    resetHSDetail: (state) => {
      state.hoSoManage = initialState.hoSoManage;
    },
  },
  extraReducers: (builder) => {
    builder
      .addCase(getHoSoManage.pending, (state) => {
        state.hoSoManageList = {
          ...state.hoSoManageList,
          loading: true,
        };
      })
      .addCase(getHoSoManage.fulfilled, (state, action) => {
        state.hoSoManageList = {
          loading: false,
          data: action.payload,
        };
      })
      .addCase(getHoSoManage.rejected, (state) => {
        state.hoSoManageList = {
          ...state.hoSoManageList,
          loading: false,
        };
      })
      .addCase(getHoSoManageById.pending, (state) => {
        state.hoSoManage = { ...state.hoSoManage, loading: true };
      })
      .addCase(getHoSoManageById.fulfilled, (state, action) => {
```

```
        state.hoSoManage = { data: action.payload, loading: false };
      })
      .addCase(getHoSoManageById.rejected, (state, action) => {
        state.hoSoManage = { ...state.hoSoManage, loading: false };
      })
      .addCase(updateHoSoManage.pending, (state) => {
        state.hoSoManageList = {
          ...state.hoSoManageList,
          loading: true,
        };
      })
      .addCase(updateHoSoManage.fulfilled, (state) => {
        state.hoSoManageList = {
          ...state.hoSoManageList,
          loading: false,
        };
      })
      .addCase(updateHoSoManage.rejected, (state) => {
        state.hoSoManageList = {
          ...state.hoSoManageList,
          loading: false,
        };
      })
      .addCase(createHoSoManage.pending, (state) => {
        state.hoSoManageList = {
          ...state.hoSoManageList,
          loading: true,
        };
      })
      .addCase(createHoSoManage.fulfilled, (state) => {
        state.hoSoManageList = {
          ...state.hoSoManageList,
          loading: false,
        };
      })
      .addCase(createHoSoManage.rejected, (state) => {
        state.hoSoManageList = {
          ...state.hoSoManageList,
          loading: false,
        };
      })
      .addCase(deleteHoSoManage.pending, (state) => {
        state.hoSoManageList = {
          ...state.hoSoManageList,
          loading: true,
        };
      })
      .addCase(deleteHoSoManage.fulfilled, (state, action) => {
        state.hoSoManageList = {
          ...state.hoSoManageList,
          loading: false,
        };
      })
      .addCase(deleteHoSoManage.rejected, (state) => {
        state.hoSoManageList = {
          ...state.hoSoManageList,
          loading: false,
        };
      });
  },
});

export const { updateFormHS, resetFormHS, resetHSDetail } =
  hoSoManageSlice.actions;

export default hoSoManageSlice.reducer;
```