```
import { createSlice, createAsyncThunk } from "@reduxjs/toolkit";
import trainingProgramService from "services/Training/TrainingProgramService";
import { cloneDeep } from "lodash";

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

export const updateTrainingProgram = createAsyncThunk(
  "trainingPrograms/updateTrainingProgram",
  async (data, { rejectWithValue, dispatch }) => {
    try {
      const { onSuccess } = data;
      const payload = cloneDeep(data);
      delete payload.onSuccess;
      const response = await trainingProgramService.Update(data);
      if (onSuccess) onSuccess(response);

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

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

export const createTrainingProgram = createAsyncThunk(
  "trainingPrograms/createTrainingProgram",
  async (data, { rejectWithValue, dispatch }) => {
    try {
      const { onSuccess } = data;
      const payload = cloneDeep(data);
      delete payload.onSuccess;
      const response = await trainingProgramService.Create(data);
      if (onSuccess) onSuccess(response.data);

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

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

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

const initialState = {
  loading: false,
  trainingProgramList: [],
  trainingProgram: {
    data: null,
    loading: false,
  },
  error: null,
  initForm: {
    name: null,
    startDate: null,
    endDate: null,
    trainingCenterId: null,
    trainingMethodId: null,
    requireExam: null,
    originalDegree: false,
    commitmentTime: null,
    totalTuitionFee: null,
    code: null,
    comment: null,
  },
};

const trainingProgramsSlice = createSlice({
  name: "trainingPrograms",
  initialState,
  reducers: {
    updateInitForm: (state, action) => {
      state.initForm = { ...state.initForm, ...action.payload };
    },
    resetInitForm: (state) => {
      state.initForm = initialState.initForm;
    },
    resetTrainingProgram: (state) => {
      state.trainingProgram = initialState.trainingProgram;
    },
  },
  extraReducers: (builder) => {
    builder
      .addCase(getTrainingPrograms.pending, (state) => {
        state.loading = true;
      })
      .addCase(getTrainingPrograms.fulfilled, (state, action) => {
        state.loading = false;
        state.trainingProgramList = action.payload;
      })
      .addCase(getTrainingPrograms.rejected, (state) => {
        state.loading = false;
      })
      .addCase(updateTrainingProgram.pending, (state) => {
        state.loading = true;
      })
      .addCase(updateTrainingProgram.fulfilled, (state) => {
        state.loading = false;
      })
      .addCase(updateTrainingProgram.rejected, (state) => {
        state.loading = false;
      })
      .addCase(createTrainingProgram.pending, (state) => {
```

```
        state.loading = true;
      })
      .addCase(createTrainingProgram.fulfilled, (state) => {
        state.loading = false;
      })
      .addCase(createTrainingProgram.rejected, (state) => {
        state.loading = false;
      })
      .addCase(deleteTrainingProgram.pending, (state) => {
        state.loading = true;
      })
      .addCase(deleteTrainingProgram.fulfilled, (state, action) => {
        state.loading = false;
      })
      .addCase(deleteTrainingProgram.rejected, (state) => {
        state.loading = false;
      })

      .addCase(getTrainingProgramById.pending, (state) => {
        state.trainingProgram = { ...state.trainingProgram, loading: true };
      })
      .addCase(getTrainingProgramById.fulfilled, (state, action) => {
        if (action.payload) {
          state.trainingProgram = { data: action.payload, loading: false };
        } else {
          state.trainingProgram = { data: null, loading: false };
        }
      })
      .addCase(getTrainingProgramById.rejected, (state, action) => {
        state.trainingProgram = { ...state.trainingProgram, loading: false };
      });
  },
});
export const { updateInitForm, resetInitForm, resetTrainingProgram } =
  trainingProgramsSlice.actions;

export default trainingProgramsSlice.reducer;
```