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

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

export const updateTrainingMethod = createAsyncThunk(
  "trainingMethods/updateTrainingMethod",
  async (data, { rejectWithValue }) => {
    try {
      const { onSuccess } = data;
      const payload = cloneDeep(data);
      delete payload.onSuccess;
      const response = await trainingMethodService.Update(data);
      if (onSuccess) onSuccess(response);
      return response.data;
    } catch (error) {
      console.error("API call failed:", error);
      return rejectWithValue(error.message || "Error");
    }
  }
);

export const createTrainingMethod = createAsyncThunk(
  "trainingMethods/createTrainingMethod",
  async (data, { rejectWithValue }) => {
    try {
      const { onSuccess } = data;
      const payload = cloneDeep(data);
      delete payload.onSuccess;
      const response = await trainingMethodService.Create(data);
      if (onSuccess) onSuccess(response);
      return response.data;
    } catch (error) {
      console.error("API call failed:", error);
      return rejectWithValue(error.message || "Error");
    }
  }
);

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

const initialState = {
  loading: false,
  trainingMethodList: [],
  trainingMethodListSetting: [],
  error: null,
```

```
};

const trainingMethodsSlice = createSlice({
  name: "trainingMethods",
  initialState,
  reducers: {},
  extraReducers: (builder) => {
    builder
      .addCase(getTrainingMethods.pending, (state) => {
        state.loading = true;
        state.error = null;
      })
      .addCase(getTrainingMethods.fulfilled, (state, action) => {
        state.loading = false;
        const payload = action.payload || [];
        const filteredPayload = payload.filter(
          (item) => item.inActive === false
        );
        state.trainingMethodList = filteredPayload;
        state.trainingMethodListSetting = [
          {
            action: "initial",
            isRequired: true,
          },
          ...payload,
        ];
      })
      .addCase(getTrainingMethods.rejected, (state, action) => {
        state.loading = false;
      })
      .addCase(updateTrainingMethod.pending, (state) => {
        state.loading = true;
      })
      .addCase(updateTrainingMethod.fulfilled, (state, action) => {
        state.loading = false;
      })
      .addCase(updateTrainingMethod.rejected, (state, action) => {
        state.loading = false;
      })
      .addCase(createTrainingMethod.pending, (state) => {
        state.loading = true;
      })
      .addCase(createTrainingMethod.fulfilled, (state, action) => {
        state.loading = false;
      })
      .addCase(createTrainingMethod.rejected, (state, action) => {
        state.loading = false;
      })
      .addCase(deleteTrainingMethod.pending, (state) => {
        state.loading = true;
      })
      .addCase(deleteTrainingMethod.fulfilled, (state, action) => {
        state.loading = false;
      })
      .addCase(deleteTrainingMethod.rejected, (state, action) => {
        state.loading = false;
      });
  },
});

export default trainingMethodsSlice.reducer;
```