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

export const getAllDocumentTypes = createAsyncThunk(
  "common/getAllDocumentTypes",
  async (_, { rejectWithValue }) => {
    try {
      const response = await SignatureProcessService.getAllDocumentTypes();
      return response.data;
    } catch (err) {
      return rejectWithValue(err.message || "Error");
    }
  }
);

export const getAllSignPositions = createAsyncThunk(
  "common/getAllSignPositions",
  async (_, { rejectWithValue }) => {
    try {
      const response = await SignatureProcessService.getAllSignPositions();
      return response.data;
    } catch (err) {
      return rejectWithValue(err.message || "Error");
    }
  }
);

export const signPositionSearchGrid = createAsyncThunk(
  "common/signPositionSearchGrid",
  async (_, { rejectWithValue }) => {
    try {
      const response = await SignatureProcessService.searchGrid();
      return response.data;
    } catch (err) {
      return rejectWithValue(err.message || "Error");
    }
  }
);

export const getAllSignatureProcess = createAsyncThunk(
  "common/getAllSignatureProcess",
  async (data, { rejectWithValue }) => {
    try {
      const response = await SignatureProcessService.getAllSignatureProcess(
        data
      );
      return response.data;
    } catch (err) {
      return rejectWithValue(err.message || "Error");
    }
  }
);

export const getSignatureProcessById = createAsyncThunk(
  "common/getSignatureProcessById",
  async (id, { rejectWithValue }) => {
    try {
      const response = await SignatureProcessService.getSignatureProcessById(
        id
      );
      return response.data;
    } catch (err) {
      return rejectWithValue(err.message || "Error");
    }
  }
);

export const createSignatureProcess = createAsyncThunk(
  "common/createSignatureProcess",
  async (data, { rejectWithValue }) => {
```

```
    try {
      const { onSuccess } = data;
      const payload = cloneDeep(data);
      delete payload.onSuccess;
      const response = await SignatureProcessService.create(payload);
      if (onSuccess) onSuccess(response);
      return response.data;
    } catch (err) {
      return rejectWithValue(err.message || "Error");
    }
  }
);

export const updateSignatureProcess = createAsyncThunk(
  "common/updateSignatureProcess",
  async (data, { rejectWithValue }) => {
    try {
      const { onSuccess } = data;
      const payload = cloneDeep(data);
      delete payload.onSuccess;
      const response = await SignatureProcessService.update(payload);
      if (onSuccess) onSuccess(response);
      return response.data;
    } catch (err) {
      return rejectWithValue(err.message || "Error");
    }
  }
);

export const deleteSignatureProcess = createAsyncThunk(
  "common/deleteSignatureProcessById",
  async (data, { rejectWithValue }) => {
    try {
      const { onSuccess, id } = data;
      const response = await SignatureProcessService.delete(id);
      if (onSuccess) onSuccess(response);
      return response.data;
    } catch (err) {
      return rejectWithValue(err.message || "Error");
    }
  }
);

export const getSignedDocument = createAsyncThunk(
  "common/getSignedDocument",
  async (data, { rejectWithValue }) => {
    try {
      const { onSuccess } = data;
      const payload = cloneDeep(data);
      delete payload.onSuccess;
      const response = await SignatureProcessService.getSignedDocument(payload);
      if (onSuccess) onSuccess(response);
      return response.data;
    } catch (err) {
      return rejectWithValue(err.message || "Error");
    }
  }
);

export const createSignedDocument = createAsyncThunk(
  "common/createSignedDocument",
  async (data, { rejectWithValue }) => {
    try {
      const { onSuccess } = data;
      const payload = cloneDeep(data);
      delete payload.onSuccess;
      const response = await SignatureProcessService.createSignedDocument(
        payload
      );
      if (onSuccess) onSuccess(response.data);
      return response.data;
    } catch (err) {
      return rejectWithValue(err.message || "Error");
    }
```

```
  }
);

export const updateSignedDocument = createAsyncThunk(
  "common/updateSignedDocument",
  async (data, { rejectWithValue }) => {
    try {
      const { onSuccess } = data;
      const payload = cloneDeep(data);
      delete payload.onSuccess;
      const response = await SignatureProcessService.updateSignedDocument(
        payload
      );
      if (onSuccess) onSuccess(response.data);
      return response.data;
    } catch (err) {
      return rejectWithValue(err.message || "Error");
    }
  }
);

export const deleteSignedDocument = createAsyncThunk(
  "common/deleteSignedDocument",
  async (data, { rejectWithValue }) => {
    try {
      const { onSuccess } = data;
      const response = await SignatureProcessService.deleteSignedDocument(data);
      if (onSuccess) onSuccess(response);
      return response.data;
    } catch (err) {
      return rejectWithValue(err.message || "Error");
    }
  }
);

export const signRemind = createAsyncThunk(
  "common/signRemind",
  async (data, { rejectWithValue }) => {
    try {
      const { onSuccess } = data;
      const payload = cloneDeep(data);
      delete payload.onSuccess;
      const response = await SignatureProcessService.signRemind(payload);
      if (onSuccess) onSuccess(response);
      return response.data;
    } catch (err) {
      return rejectWithValue(err.message || "Error");
    }
  }
);

export const signReject = createAsyncThunk(
  "common/signReject",
  async (data, { rejectWithValue }) => {
    try {
      const { onSuccess } = data;
      const payload = cloneDeep(data);
      delete payload.onSuccess;
      const response = await SignatureProcessService.signReject(payload);
      if (onSuccess) onSuccess(response.data);
      return response.data;
    } catch (err) {
      return rejectWithValue(err.message || "Error");
    }
  }
);

export const createSignPosition = createAsyncThunk(
  "common/createSignPosition",
  async (data, { rejectWithValue }) => {
    try {
      const { onSuccess } = data;
      const payload = cloneDeep(data);
      delete payload.onSuccess;
```

```
      const response = await SignatureProcessService.createSignPosition(
        payload
      );
      if (onSuccess) onSuccess(response.data);
      return response.data;
    } catch (err) {
      return rejectWithValue(err.message || "Error");
    }
  }
);

export const updateSignPosition = createAsyncThunk(
  "common/updateSignPosition",
  async (data, { rejectWithValue }) => {
    try {
      const { onSuccess } = data;
      const payload = cloneDeep(data);
      delete payload.onSuccess;
      const response = await SignatureProcessService.updateSignPosition(
        payload
      );
      if (onSuccess) onSuccess(response.data);
      return response.data;
    } catch (err) {
      return rejectWithValue(err.message || "Error");
    }
  }
);

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

//SIGN_REPORT
export const getSignReport = createAsyncThunk(
  "signatureProcess/getSignReport",
  async (data, { rejectWithValue }) => {
    try {
      const response = await SignatureProcessService.GetSignReport(data);
      return response.data;
    } catch (err) {
      console.error("API call failed:", err);
      return rejectWithValue(err.message || "Error");
    }
  }
);

//END SIGN_REPORT

const initialState = {
  loading: false,
  signatureProcessList: [],
  signatureProcessDetail: {},
  documentTypeList: [],
  signPositionList: [],
  signReportList: {
    data: [],
    loading: false,
  },
};

export const signatureProcessSlice = createSlice({
  name: "signatureProcess",
```

```
  initialState,
  reducers: {
    showLoading: (state) => {
      state.loading = true;
    },
  },
  extraReducers: (builder) => {
    builder
      .addCase(getAllDocumentTypes.pending, (state) => {
        state.loading = true;
      })
      .addCase(getAllDocumentTypes.fulfilled, (state, action) => {
        state.loading = false;
        state.documentTypeList = action.payload;
      })
      .addCase(getAllDocumentTypes.rejected, (state) => {
        state.loading = true;
      })
      .addCase(getAllSignPositions.pending, (state) => {
        state.loading = true;
      })
      .addCase(getAllSignPositions.fulfilled, (state, action) => {
        state.loading = false;
        state.signPositionList = action.payload;
      })
      .addCase(getAllSignPositions.rejected, (state) => {
        state.loading = true;
      })
      .addCase(signPositionSearchGrid.pending, (state) => {
        state.loading = true;
      })
      .addCase(signPositionSearchGrid.fulfilled, (state, action) => {
        state.loading = false;
        state.signPositionList = action.payload;
      })
      .addCase(signPositionSearchGrid.rejected, (state) => {
        state.loading = true;
      })
      .addCase(getAllSignatureProcess.pending, (state) => {
        state.loading = true;
      })
      .addCase(getAllSignatureProcess.fulfilled, (state, action) => {
        state.loading = false;
        state.signatureProcessList = action.payload;
      })
      .addCase(getAllSignatureProcess.rejected, (state, action) => {
        state.loading = false;
      })
      .addCase(getSignatureProcessById.pending, (state) => {
        state.loading = true;
      })
      .addCase(getSignatureProcessById.fulfilled, (state, action) => {
        state.loading = false;
        state.signatureProcessDetail = action.payload;
      })
      .addCase(getSignatureProcessById.rejected, (state, action) => {
        state.loading = false;
      })
      .addCase(createSignatureProcess.pending, (state) => {
        state.loading = true;
      })
      .addCase(createSignatureProcess.fulfilled, (state, action) => {
        state.loading = false;
      })
      .addCase(createSignatureProcess.rejected, (state, action) => {
        state.loading = false;
      })
      .addCase(updateSignatureProcess.pending, (state) => {
        state.loading = true;
      })
      .addCase(updateSignatureProcess.fulfilled, (state, action) => {
        state.loading = false;
      })
      .addCase(updateSignatureProcess.rejected, (state, action) => {
```

```
  state.loading = false;
})
.addCase(deleteSignatureProcess.pending, (state) => {
  state.loading = true;
})
.addCase(deleteSignatureProcess.fulfilled, (state, action) => {
  state.loading = false;
})
.addCase(deleteSignatureProcess.rejected, (state, action) => {
  state.loading = false;
})
.addCase(getSignedDocument.pending, (state) => {
  state.loading = true;
})
.addCase(getSignedDocument.fulfilled, (state, action) => {
  state.loading = false;
})
.addCase(getSignedDocument.rejected, (state, action) => {
  state.loading = false;
})
.addCase(createSignedDocument.pending, (state) => {
  state.loading = true;
})
.addCase(createSignedDocument.fulfilled, (state, action) => {
  state.loading = false;
})
.addCase(createSignedDocument.rejected, (state, action) => {
  state.loading = false;
})
.addCase(updateSignedDocument.pending, (state) => {
  state.loading = true;
})
.addCase(updateSignedDocument.fulfilled, (state, action) => {
  state.loading = false;
})
.addCase(updateSignedDocument.rejected, (state, action) => {
  state.loading = false;
})
.addCase(deleteSignedDocument.pending, (state) => {
  state.loading = true;
})
.addCase(deleteSignedDocument.fulfilled, (state, action) => {
  state.loading = false;
})
.addCase(deleteSignedDocument.rejected, (state, action) => {
  state.loading = false;
})
.addCase(signRemind.pending, (state) => {
  state.loading = true;
})
.addCase(signRemind.fulfilled, (state, action) => {
  state.loading = false;
})
.addCase(signRemind.rejected, (state, action) => {
  state.loading = false;
})
.addCase(signReject.pending, (state) => {
  state.loading = true;
})
.addCase(signReject.fulfilled, (state, action) => {
  state.loading = false;
})
.addCase(signReject.rejected, (state, action) => {
  state.loading = false;
})
.addCase(createSignPosition.pending, (state) => {
  state.loading = true;
})
.addCase(createSignPosition.fulfilled, (state, action) => {
  state.loading = false;
})
.addCase(createSignPosition.rejected, (state, action) => {
  state.loading = false;
})
```

```
      .addCase(updateSignPosition.pending, (state) => {
        state.loading = true;
      })
      .addCase(updateSignPosition.fulfilled, (state, action) => {
        state.loading = false;
      })
      .addCase(updateSignPosition.rejected, (state, action) => {
        state.loading = false;
      })
      .addCase(deleteSignPosition.pending, (state) => {
        state.loading = true;
      })
      .addCase(deleteSignPosition.fulfilled, (state, action) => {
        state.loading = false;
      })
      .addCase(deleteSignPosition.rejected, (state, action) => {
        state.loading = false;
      })
      //SIGN-REPORT
      .addCase(getSignReport.pending, (state) => {
        state.signReportList = {
          ...state.signReportList,
          loading: true,
        };
      })
      .addCase(getSignReport.fulfilled, (state, action) => {
        state.signReportList = {
          loading: false,
          data: action.payload,
        };
      })
      .addCase(getSignReport.rejected, (state) => {
        state.signReportList = {
          ...state.signReportList,
          loading: false,
        };
      });
  },
});

export const { showLoading, setHisInfo } = signatureProcessSlice.actions;

export default signatureProcessSlice.reducer;
```