```javascript
import { createSlice, createAsyncThunk } from "@reduxjs/toolkit";
import { cloneDeep } from "lodash";
import BangCongService from "services/BangCongService";
import { initThietLapBangCong } from "views/app-views/hrm/bangcongBaoHiem/thietLapBangCong";

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

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

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

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

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

export const deleteDsBangCong = createAsyncThunk(
  "bangCong/deleteDsBangCong",
  async (data, { rejectWithValue }) => {
    try {
      const { onSuccess, id } = data;
      const response = await BangCongService.deleteBangCong(id);
      if (onSuccess) onSuccess(response.data);
      return response.data;
    } catch (err) {
      return rejectWithValue(err.message || "Error");
    }
  }
);

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

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

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

export const getBangCongSystem = createAsyncThunk(
  "bangCong/getBangCongSystem",
  async (branchId, { rejectWithValue }) => {
    try {
      const response = await BangCongService.getBangCongSystem(branchId);
      return response.data;
    } catch (err) {
      return rejectWithValue(err.message || "Error");
    }
  }
);

export const DeleteBangCongSystem = createAsyncThunk(
  "bangCong/DeleteBangCongSystem",
  async (data, { rejectWithValue }) => {
    try {
      const { onSuccess, id } = data;
      const response = await BangCongService.delBangCongSystem(id);
      if (onSuccess) onSuccess(response.data);
      return response.data;
    } catch (err) {
      return rejectWithValue(err.message || "Error");
    }
  }
);

export const CopyBangCong = createAsyncThunk(
  "bangCong/CopyBangCong",
  async (data, { rejectWithValue }) => {
    try {
      const { onSuccess, id } = data;
      const response = await BangCongService.copyBangCong(id);
      if (onSuccess) onSuccess(response.data);
      return response.data;
    } catch (err) {
      return rejectWithValue(err.message || "Error");
    }
  }
);

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

export const deleteNhanVienById = createAsyncThunk(
  "bangCong/deleteNhanVienById",
  async (data, { rejectWithValue }) => {
    try {
      const { onSuccess, id } = data;
      const response = await BangCongService.delNhanVienById(id);
      if (onSuccess) onSuccess(response.data);
      return response.data;
    } catch (err) {
      return rejectWithValue(err.message || "Error");
    }
  }
);

export const getBangCongTongHopNam = createAsyncThunk(
  "bangCong/getBangCongTongHopNam",
  async (branchId, { rejectWithValue }) => {
    try {
      const response = await BangCongService.getTongHopNam(branchId);
      return response.data;
    } catch (err) {
      return rejectWithValue(err.message || "Error");
    }
  }
);
```

```
export const getBangCongTongHopByNam = createAsyncThunk(
  "bangCong/getBangCongTongHopByNam",
  async (data, { rejectWithValue }) => {
    try {
      const response = await BangCongService.getTongHopNamByNam(data);
      return response.data;
    } catch (err) {
      return rejectWithValue(err.message || "Error");
    }
  }
);

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

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

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

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

export const upSertBangCongHisAsync = createAsyncThunk(
  "bangCong/upSertBangCongHisAsync",
  async (data, { rejectWithValue }) => {
    try {
      const { onSuccess, bangCongList, reqType } = data;
      const response = await BangCongService.upSertbangCongHisAsync(
        bangCongList,
        reqType
      );
      if (onSuccess) onSuccess(response.data);
      return response.data;
    } catch (err) {
      return rejectWithValue(err.message || "Error");
    }
  }
);

export const bangCongFormKhieuNaiChangeState = createAsyncThunk(
  "bangCong/bangCongFormKhieuNaiChangeState",
  async (data, { rejectWithValue }) => {
    try {
      const { onSuccess, id } = data;
      const response = await BangCongService.BangCongFormKhieuNaiChangeState(
        id
      );
      if (onSuccess) onSuccess(response.data);
      return response.data;
    } catch (err) {
      return rejectWithValue(err.message || "Error");
    }
  }
);

const initialState = {
  loading: false,
  bangCong: {},
  bangCongDetail: {},
  ChiTietBCList: [],
  ngayLamViecList: [],
  bangCongCate: [],
  thietLap: [],
  thietLapCongList: [],
  tongHopNamList: [],
  tongHopNamDetail: {},
  nhanVienList: [],
  nhanVienHisAsync: [],
  bangCongFormKhieuNaiList: [],
};

export const bangCongSlice = createSlice({
  name: "bangCong",
  initialState,
  reducers: {
    showLoading: (state) => {
      state.loading = true;
    },
    setChiTietBc: (state, action) => {
      state.ChiTietBCList = action.payload;
    },
    setNhanVienHis: (state, action) => {
      state.nhanVienHisAsync = action.payload;
    },
    setBangCongDetail: (state, action) => {
      state.bangCongDetail = action.payload;
    },
    setThietLap: (state, action) => {
      state.thietLap = action.payload;
    },
  },
  extraReducers: (builder) => {
    builder
      .addCase(searchDsBangCong.fulfilled, (state, action) => {
        state.loading = false;
        state.bangCong = action.payload;
      })
      .addCase(searchDsBangCong.rejected, (state, action) => {
        state.loading = false;
      })
      .addCase(searchDsBangCong.pending, (state) => {
        state.loading = true;
      })

      .addCase(getByIdDsBangCong.fulfilled, (state, action) => {
        state.loading = false;
        state.bangCongDetail = action.payload;
      })
      .addCase(getByIdDsBangCong.rejected, (state, action) => {
        state.loading = false;
      })
      .addCase(getByIdDsBangCong.pending, (state) => {
        state.loading = true;
      })

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

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

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

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

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

      .addCase(searchChiTietBangCong.fulfilled, (state, action) => {
        state.loading = false;
        state.ChiTietBCList = action.payload.data;
        state.ngayLamViecList = action.payload.ngayLamViecList;
        state.bangCongCate = action.payload;
        state.thietLapCongList = action.payload.thietLapCongList;
      })
      .addCase(searchChiTietBangCong.rejected, (state, action) => {
        state.loading = false;
      })
      .addCase(searchChiTietBangCong.pending, (state) => {
        state.loading = true;
      })

      .addCase(getBangCongSystem.fulfilled, (state, action) => {
        state.loading = false;
        state.thietLap = [initThietLapBangCong, ...action.payload];
      })
      .addCase(getBangCongSystem.rejected, (state, action) => {
        state.loading = false;
      })
      .addCase(getBangCongSystem.pending, (state) => {
        state.loading = true;
      })

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

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

      .addCase(getBangCongTongHopNam.fulfilled, (state, action) => {
        state.loading = false;
        state.tongHopNamList = action.payload;
      })
      .addCase(getBangCongTongHopNam.rejected, (state, action) => {
        state.loading = false;
      })
      .addCase(getBangCongTongHopNam.pending, (state) => {
        state.loading = true;
      })

      .addCase(getBangCongTongHopByNam.fulfilled, (state, action) => {
        state.loading = false;
        state.tongHopNamDetail = action.payload;
      })
      .addCase(getBangCongTongHopByNam.rejected, (state, action) => {
        state.loading = false;
      })
      .addCase(getBangCongTongHopByNam.pending, (state) => {
        state.loading = true;
      })

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

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

      .addCase(checkBangCongNhanVien.fulfilled, (state, action) => {
        state.loading = false;
        state.nhanVienList = action.payload.data;
      })
      .addCase(checkBangCongNhanVien.rejected, (state, action) => {
        state.loading = false;
      })
      .addCase(checkBangCongNhanVien.pending, (state) => {
        state.loading = true;
      })
      .addCase(bangCongHisAsync.fulfilled, (state, action) => {
        state.loading = false;
        state.nhanVienHisAsync = action.payload.data;
      })
      .addCase(bangCongHisAsync.rejected, (state, action) => {
        state.loading = false;
      })
      .addCase(bangCongHisAsync.pending, (state) => {
        state.loading = true;
      })
      .addCase(upSertBangCongHisAsync.fulfilled, (state, action) => {
        state.loading = false;
      })
      .addCase(upSertBangCongHisAsync.rejected, (state, action) => {
        state.loading = false;
      })
      .addCase(upSertBangCongHisAsync.pending, (state) => {
        state.loading = true;
      })
      .addCase(deleteNhanVienById.fulfilled, (state, action) => {
        state.loading = false;
      })
      .addCase(deleteNhanVienById.rejected, (state, action) => {
        state.loading = false;
      })
      .addCase(deleteNhanVienById.pending, (state) => {
        state.loading = true;
      })

      .addCase(searchBangCongFormKhieuNai.fulfilled, (state, action) => {
        state.loading = false;
        state.bangCongFormKhieuNaiList = action.payload;
      })
      .addCase(searchBangCongFormKhieuNai.rejected, (state, action) => {
        state.loading = false;
      })
      .addCase(searchBangCongFormKhieuNai.pending, (state) => {
        state.loading = true;
      })

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

export const {
  showLoading,
  setChiTietBc,
  setBangCongDetail,
  setNhanVienHis,
  setThietLap,
} = bangCongSlice.actions;

export default bangCongSlice.reducer;
```