```
const RANK_AND_FILE = 1;
const SUPERVISOR = 2;
const MANAGER = 3;

const EXEMPT = 1;
const NONEXEMPT = 2;

const HOURLY = 1;
const MONTHLY = 2;


export default {
    name: 'ContractWorkflowPage',

    components: {
        Page,
        Button,
        BackButton,
        ContractConfig,
        EmployeeFirstContract,
        // CreateScheduleModal,
        SignatureModal,
        BaseLoading,
        AlertBanner,
    },

    mixins: [
        HideFooterMixin,
        CurrencyMixin,
        FieldValidatorMixin,
        UploadFileMixin,
    ],

    provide() {
        return {
            getAddContract: () => this.isAddContract,
            getContractId: () => this.prevContract.pk
        };
    },
```

```
            },

            showAddContractButton() {
                return !this.isForRelease && !this.isLoading &&
this.isOrgExecutive;
            },

            contractWorkflowInfo() {
                const applicantName = `${this.applicant?.first_name}
${this.applicant?.last_name}`;
                return this.isOrgExecutive
                    ? `Edit contract workflow for ${applicantName}.`
                    : `Contact HR to edit contract workflow for
${applicantName}.`;
            }
        },

        async created() {
            this.isLoading = true;
            this.applicant = await
this.getJobOfferApplicantDetails(parseInt(this.$route.query.applicant_id))
;
            if (this.isForRelease) {
                this.nextContracts.push({ pk:
parseInt(this.$route.query.contract_id) });
            } else {
                this.nextContracts = await this.fetchEmploymentContracts({
                    employment_id:
parseInt(this.$route.params.employment_id)
                });
                this.isAddContract = this.nextContracts.length > 1;
                this.prevContract = this.nextContracts[0];
                setTimeout(() => {
                    this.isLoading = false;
                }, 300);
            }
            // TODO Investigate if editing/adding of schedules is still
needed here
            // await this.fetchScheduleGroups();
        },
```

```
                    compensation_type:
this.getCompensationTypeValue(employmentRule.type),
                    content: JSON.stringify(jobOffer.contractContent),
                    start_date: jobOffer.startDate,
                    job_position: jobOffer.jobTitle,
                    office_location: this.prevContract.office_location?.pk,
                    salary: this.getSalaryValue(jobOffer, employmentRule),
                    currency: jobOffer.currency,
                    applicant_name: `${this.applicant.first_name}
${this.applicant.last_name}`,
                    name: config.name,
                    release_date: config.contractPolicy.release_date,
                    related_job_offer:
parseInt(this.prevContract.related_job_offer.pk),
                    signature_location: this.signatureDetails.location,
                    signature_uuid: uuid,
                    is_additional_main_contract: this.isAddContract
                };
                let response;
                if (config.contractPk) {
                    data.pk = config.contractPk;
                    response = await this.updateContract(data);
                    this.contractEdited = true;
                } else {
                    response = await this.createEmploymentContract(data);
                    this.nextContracts.splice(parseInt(index), 1);
                    this.nextContracts.push(response);
                    this.contractSaved = true;
                }

                this.isSubmitting = false;
                this.disableAddContractButton = false;
                setTimeout(() => {
                    this.contractSaved = false;
                    this.contractEdited = false;
                }, 3000);
                return data;
            },

            async deleteContractHandler(index) {
```