```vue
<template>
  <Page>
    <template v-if="isLoading">
      <div class="information">
        <BaseLoading
          height="30px"
          width="200px"
          border-radius="4px"
        />
        <BaseLoading
          height="24px"
          width="100px"
          border-radius="4px"
        />
      </div>
    </template>
    <template v-else>
      <div class="header-wrapper">
        <BackButton :route="previousRoute" />
        <div class="header-description">
          <p class="p-h1 c-primary">
            Contract Workflow
          </p>
        </div>
      </div>
    </template>
    <AlertBanner
      v-if="!isLoading"
      icon="information-outline"
      :label="contractWorkflowInfo"
      color="blue"
    />
    <div class="body">
      <template v-if="isLoading">
        <BaseLoading
          height="600px"
          width="1200px"
          border-radius="4px"
        />
        <div class="add-template">
```

```
                    <BaseLoading
                        height="40px"
                        width="200px"
                        border-radius="4px"
                    />
                </div>
            </template>
            <template v-else>
                <template
                    v-for="(contract, index) in nextContracts"
                >
                    <EmployeeFirstContract
                        v-if="contract.pk && (contract.date_signed ||
contract.is_active)"
                        :key="contract.pk || contract.contractPk || index"
                        :contract="contract"
                        :contract-policy="contract.contractPolicy"
                    />
                    <ContractConfig
                        v-else
                        :key="contract.pk || contract.contractPk"
                        v-model="nextContracts[index]"
                        :is-for-release="isForRelease"
                        :is-editable="true"
                        :applicant="applicant"
                        :start-date="nextContracts[0].start_date"
                        :is-saving-contract="isSubmitting"
                        :contract-saved="contractSaved"
                        :contract-edited="contractEdited"
                        @save-contract="openSignatureModal(index)"
                        @delete-contract="deleteContractHandler(index)"
                    />
                </template>
                <div class="add-template">
                    <Button
                        v-if="showAddContractButton"
                        :disabled="disableAddContractButton"
                        class="large border primary bold"
                        icon="plus-circle"
                        @click="addContractHandler()"
```

```
                    >
                        Create another contract
                    </Button>
                </div>
            </template>
        </div>
        <!-- <CreateScheduleModal
            v-if="createSchedule"
            :id="-1"
            :company-id="companyId"
            @close-modal="createSchedule = false"
            @save-schedule="saveSchedule"
        /> -->
        <SignatureModal
            v-if="isSignatureOpen"
            @close="closeSignatureModal"
            @confirm-signature="createContract"
       />
    </Page>
</template>

<script>
    import Page from 'source/components/_generics/layouts/Page.vue';
    import Button from 'source/components/_generics/Button.vue';
    import BackButton from 'source/components/_generics/BackButton.vue';
    import ContractConfig from
'source/pages/employers/contracts/ContractConfig.vue';
    import EmployeeFirstContract from
'source/components/employers/employees/EmployeeFirstContract.vue';
    // import CreateScheduleModal from
'source/components/employers/work-schedule/CreateScheduleModal.vue';
    import HideFooterMixin from 'source/mixins/HideFooterMixin.js';
    import CurrencyMixin from 'source/mixins/CurrencyMixin.js';
    import FieldValidatorMixin from 'source/mixins/FieldValidatorMixin.js';
    import { mapActions, mapState } from 'vuex';
    import SignatureModal from
'source/components/_generics/SignatureModal.vue';
    import UploadFileMixin from 'source/mixins/UploadFileMixin.js';
    import BaseLoading from 'source/components/_generics/BaseLoading.vue';
    import AlertBanner from 'source/components/_generics/AlertBanner.vue';
```

```
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
        };
    },
```

```
    data() {
        return {
            isCampaignPolicyOpen: false,
            applicant: null,
            nextContracts: [],
            isSubmitting: false,
            isLoading: false,
            // createSchedule: false,
            // jobScheduleGroups: [],
            contractSaved: false,
            contractEdited: false,
            isAddContract: false,
            disableAddContractButton: false,
            prevContract: [],
            isSignatureOpen: false,
            selectedIndex: null,
            signatureDetails: null,
        };
    },

    computed: {
        ...mapState('auth', ['user']),

        previousRoute() {
            return {
                name: 'staff-employee-details'
            };
        },

        isForRelease() {
            return !!this.$route.query.contract_id;
        },

        companyId() {
            return this.$route.params.company_id;
        },

        isOrgExecutive() {
            return this.user?.user_profile?.is_organizational_exec;
```

```
            },

            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();
        },
```

```
      methods: {
          ...mapActions('jobOffer', ['getJobOfferApplicantDetails',
'fetchJobOfferAmazonFormData']),
          ...mapActions('employers', ['updateOrCreateEmploymentRule',
'fetchJobScheduleGroups']),
          ...mapActions('contract', [
              'fetchEmploymentContracts',
              'createEmploymentContract',
              'deleteContract',
              'updateContract',
              'createContractPolicy'
          ]),

          async addContractHandler() {
              this.disableAddContractButton = true;
              this.isAddContract = true;
              this.nextContracts.push({});
          },

          // TODO Investigate if editing/adding of schedules is still
needed here
          // async saveSchedule(name) {
          //     this.createSchedule = false;
          //     await this.fetchScheduleGroups();
          //     this.jobOffer.schedule = this.jobScheduleGroups.find(s
=> s.name == name);
          // },

          // async fetchScheduleGroups() {
          //     let filters = {
          //         'company_id': this.companyId,
          //         'workforce_id': this.$route.params.workforce_id,
          //         'top': -1
          //     };

          //     this.fetchJobScheduleGroups(filters).then(response => {
          //         this.jobScheduleGroups = response.objects;
          //     }).catch(err => {
          //         console.error(err);
```

```
            //          this.jobScheduleGroups = [];
            //      });
            // },

            async createContract(signature) {
                this.isSubmitting = true;
                this.signatureDetails = signature;
                this.isSignatureOpen = false;

                let contractConfig =
this.nextContracts[this.selectedIndex];

                let cleanedEmploymentRuleData =
this.cleanEmploymentRuleData(contractConfig.employmentRule);
                let response = await
this.updateOrCreateEmploymentRule(cleanedEmploymentRuleData);

                response.error
                    ? this.isSubmitting = false
                    : this.saveEmploymentContract(
                        response.employment_rule_pk,
                        contractConfig,
                        this.selectedIndex);
            },

            cleanEmploymentRuleData(employmentRuleData) {
                let allowAfterShiftOvertime =
employmentRuleData.allowAfterShiftOvertime;

                let data = {
                    name: `${this.applicant.first_name}
${this.applicant.last_name}`,
                    position:
this.getJobPositionValue(employmentRuleData.role),
                    employee_type:
this.getEmployeeTypeValue(employmentRuleData.type),
                    compensation_type:
this.getCompensationTypeValue(employmentRuleData.payableType),
                    can_overtime: employmentRuleData.allowOvertime === 'No'
? 'False' : 'True',
```

```
                    overtime_rate: 1 +
parseFloat(employmentRuleData.overtimeRate)/100 || 0,
                    can_file_overtime_after_shift: allowAfterShiftOvertime
=== 'No' ? 'False' : 'True'
                };

                return data;
            },

            async saveEmploymentContract(employmentRuleID, config, index) {
                let employmentRule = config.employmentRule;
                let jobOffer = config.jobOffer;
                let policy = config.contractPolicy;

                let contractPolicyPk = policy?.pk;

                if (!contractPolicyPk) {
                    let response = await this.createContractPolicy({
                        company_id: this.companyId,
                        ...policy
                    });
                    contractPolicyPk = response.pk;
                }

                let uuid = await new Promise((resolve, reject) => {
                    this.signatureDetails.signature.save(async (blob) =>{
                        resolve(await this.upload(blob,
'fetchJobOfferAmazonFormData'));
                    });
                });

                let data = {
                    employment_rule: parseInt(employmentRuleID),
                    employment_policy:
parseInt(config.employmentPolicy.pk),
                    schedule: parseInt(jobOffer.schedule.pk),
                    contract_policy: parseInt(contractPolicyPk),
                    employment: parseInt(this.$route.params.employment_id),
                    role: this.getJobPositionValue(employmentRule.role),
```

```
                    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) {
```

```
                this.disableAddContractButton = false;
                let config = this.nextContracts[index];

                let data = await {
                    employment_rule: parseInt(config.employmentRule.pk),
                    employment_policy:
parseInt(config.employmentPolicy.pk),
                    schedule: parseInt(config.jobOffer.schedule.pk),
                    contract_policy: parseInt(config.contractPolicy.pk),
                    employment: parseInt(this.$route.params.employment_id),
                    role:
this.getJobPositionValue(config.employmentRule.role),
                    content:
JSON.stringify(config.jobOffer.contractContent),
                    start_date: config.jobOffer.startDate,
                    job_position: config.jobOffer.jobTitle,
                    currency: config.jobOffer.currency,
                    salary: this.getSalaryValue(config.jobOffer,
config.employmentRule),
                    compensation_type:
this.getCompensationTypeValue(config.employmentRule.type),
                    applicant_name: this.applicant.name,
                    name: config.name,
                    release_date: config.contractPolicy.release_date,
                    is_additional_main_contract: true,
                };
                if (config.contractPk) {
                    data.pk = config.contractPk;
                    await this.deleteContract(data);
                }

                this.nextContracts.splice(parseInt(index), 1);
            },

            getJobPositionValue(position) {
                if (position === 'Manager') {
                    return MANAGER;
                } else if (position === 'Supervisor') {
                    return SUPERVISOR;
                } else {
```

```
                return RANK_AND_FILE;
            }
        },

        getEmployeeTypeValue(type) {
            if (type === 'Exempt') {
                return EXEMPT;
            } else {
                return NONEXEMPT;
            }
        },

        getCompensationTypeValue(type) {
            if (type === 'Wage') {
                return HOURLY;
            } else {
                return MONTHLY;
            }
        },

        /**
         * retains value if wage / hourly
         * divides value to 12 if salary since yearly salary is being
passed
         * returns salary value in USD
         */
        getSalaryValue(jobOffer, employmentRule) {
            let salaryAmountInSelectedCurrency = 0;

            if (employmentRule.payableType === 'Wage') {
                salaryAmountInSelectedCurrency =
parseFloat(jobOffer.compensationValue);
            } else {
                salaryAmountInSelectedCurrency =
parseFloat(jobOffer.compensationValue) / 12;
            }

            let compensationAmount =
salaryAmountInSelectedCurrency.toString();
```

```
                return compensationAmount;
            },

            closeSignatureModal() {
                this.isSignatureOpen = false;
                this.isSubmitting = false;
            },

            openSignatureModal(index) {
                this.isSubmitting = true;
                this.isSignatureOpen = true;
                this.selectedIndex = index;
            },
        }
    };
</script>

<style lang="scss" scoped>
    @import 'source/stylesheets/sass/abstract/bpo-variables.scss';

    .page-view {
        display: block;
        height: calc(100vh - 40px);
        position: relative;
        width: 1320px;
        max-width: calc(100% - 48px);
        margin: auto;
        align-items: center;

        .header-wrapper {
            display: flex;
            gap: 4px;
            margin-bottom: 8px;
            background: $white;
            top: 49px;
            z-index: 10;

            .header-description {
                align-self: center;
            }
```

```
    }

    .policy-chooser {
        height: 56px;
        display: flex;
        align-items: center;
        padding: 0 8px 0 16px;
        border-radius: $corner-radius;
        border: 1px solid $quaternary;
        background: $white;
        margin-bottom: 16px;

        p {
            flex: 1;
            @include clamp-line(1);
            padding-right: 8px;
        }
    }

    .information {
        @include flex-column();
        gap: 4px;
    }

    .body {
        margin-top: 8px;

        .add-template {
            margin-bottom: 24px;
            text-align: left;
            position: relative;
        }

        :deep(.config) {
            min-height: 0;
        }
    }

    @media screen and (max-width: 1400px) {
        width: 100%;
```

```
        }
    }
</style>
```