```
<template>
   <Page>
       <div class="header-wrapper">
           <BackButton :route="previousRoute" />
           <div class="header-description">
               <p class="p-h1 c-primary">
                   Send Job Offer
               </p>
               <p class="t-sm c-tertiary">
                   to {{ applicant ? applicant.first_name : '' }}
               </p>
           </div>
           <Button
               :disabled="isSubmitting"
               class="primary"
               @click="hireApplicant"
           >
               Submit
           </Button>
       </div>
       <BaseAlert
           v-if="errorMessage"
           class="error"
           icon="alert-circle-outline"
       >
           {{ errorMessage }}
       </BaseAlert>
       <div class="body">
           <template>
               <ContractConfig
                   v-if="applicant"
                   ref="config"
                   v-model="contractConfig"
                   :applicant="applicant"
                   :company="companies[0]"
                   :is-for-release="true"
                   :job-schedule-groups="jobScheduleGroups"
                   @save-contract="hireApplicant(false)"
                   @create-schedule="createSchedule = true"
                   @send-contract-name="recieveContractName"
```

```
              />
           </template>
       </div>
       <CreateScheduleModal
           v-if="createSchedule"
           :id="-1"
           :company-id="companies[0].pk"
           @close-modal="createSchedule = false"
           @save-schedule="saveSchedule"
      />
   </Page>
</template>

<script>
   import { mapActions, mapState, mapGetters } from 'vuex';
   import Page from 'source/components/_generics/layouts/Page.vue';
   import BaseAlert from 'source/components/_generics/BaseAlert.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 CreateScheduleModal from
'source/components/employers/work-schedule/CreateScheduleModal.vue';
   import HideFooterMixin from 'source/mixins/HideFooterMixin.js';
   import FieldValidatorMixin from 'source/mixins/FieldValidatorMixin.js';
   import CurrencyMixin from 'source/mixins/CurrencyMixin.js';
   import TimezoneMixin from 'source/mixins/TimezoneMixin.js';

   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: 'SendJobOfferPage',
       components: {
           Page,
           Button,
           BaseAlert,
           BackButton,
           ContractConfig,
           CreateScheduleModal
       },

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

       props: {
           workforce_id: {
               type: [String, Number],
               required: true
           }
       },

       data() {
           return {
               isCampaignPolicyOpen: false,
               applicant: null,
               selectedPolicy: null,
               contractConfig: {},
               errorMessage: '',
               isSubmitting: false,
               createSchedule: false,
               jobScheduleGroups: [],
               selectedContractName: null
           };
       },

       computed: {
           ...mapState('employers', ['company']),
           ...mapGetters('auth', [ 'companies']),

           previousRoute() {
               return {
                   name: 'recruitment-posting',
                   params: {
                       workforce_id: this.workforce_id,
                       job_posting_pk: this.$route.params.job_posting_pk
                   }
               };
           }
       },

       async created() {
           if (!this.$route?.query?.applicant_id) {
               this.$router.replace({
                   name: 'recruitment-posting',
                   params: this.$route.params
               });
               return;
           }
           this.getJobOfferApplicantDetails(
               parseInt(this.$route.query.applicant_id))
               .then(response => {
                   this.applicant = response;
                   return this.fetchScheduleGroups();
               })
               .then(() => {})
               .catch(e => {
                   console.error('Error in getJobOfferApplicantDetails',
e);
               });
       },

       methods: {
           ...mapActions('jobOffer', ['getJobOfferApplicantDetails',
'createJobOffer']),
           ...mapActions('contract', ['createContractPolicy']),
           ...mapActions('employers', ['updateOrCreateEmploymentRule',
'fetchJobScheduleGroups']),
           saveSchedule(name) {
               this.createSchedule = false;
               this.fetchScheduleGroups();
               this.contractConfig.jobOffer.schedule =
this.jobScheduleGroups.find(s => s.name == name);
           },

           recieveContractName (contractName) {
               this.selectedContractName = contractName;
           },

           async fetchScheduleGroups() {
               let filters = {
                   'company_id': this.companies[0].pk,
                   'top': -1
               };
               let jobScheduleGroupData = await
this.fetchJobScheduleGroups(filters);
               this.jobScheduleGroups = jobScheduleGroupData.objects;
           },


           async hireApplicant(validate=true) {
               this.errorMessage = '';
               if (validate) {
                   this.$refs.config.saveContractHandler();
               } else {
                   this.isSubmitting = true;

                   let cleanedEmploymentRuleData =
this.cleanEmploymentRuleData();

this.updateOrCreateEmploymentRule(cleanedEmploymentRuleData)
                       .then((response) => {
                           if (response.error) {
                               this.errorMessage = response.error;
                           } else {
                               this.cleanAndValidateJobOfferData(
                                   response.employment_rule_pk,
                                   this.contractConfig.contractPolicy
                               );
                           }
                       }).catch((e) => {
                           this.errorMessage = 'Sorry, there was an error
submiting form';
                       }).finally(() => {
                           this.isSubmitting = false;
                       });
                   }
               },

               cleanEmploymentRuleData() {
                   let employmentRuleData =
this.contractConfig.employmentRule;
                   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 cleanAndValidateJobOfferData(employmentRuleID,
policyData) {
```

```
            let employmentRule = this.contractConfig.employmentRule;
            let jobOffer = this.contractConfig.jobOffer;

            let contractPolicyPk = policyData.pk;

            if (!contractPolicyPk) {
                const policyPk = this.getContractPolicyPk({
                    company_id: this.companies[0].pk,
                    ...policyData
                });
                if (!policyPk) return;
                contractPolicyPk = policyPk;
            }

            let startDate = this.getMomentInstance(
                jobOffer.startDate, this.applicant.region);

            let data = await {
                employment_rule_id: parseInt(employmentRuleID),
                schedule_group_id: parseInt(jobOffer.schedule.pk),
                applicant_id: parseInt(this.applicant.pk),
                employment_policy_id:
parseInt(this.contractConfig.employmentPolicy.pk),
                job_posting_id:
parseInt(this.$route.params.job_posting_pk),
                workforce_id: parseInt(this.workforce_id),
                company_id: parseInt(this.companies[0].pk),
                job_position: jobOffer.jobTitle,
                role: this.getJobPositionValue(employmentRule.role),
                start_date: startDate.format('YYYY-MM-DD HH:mm:ssZZ'),
                salary: await this.getSalaryValue(jobOffer),
                currency: jobOffer.currency,
                contract_content:
JSON.stringify(jobOffer.contractContent),
                contract_policy_id: parseInt(contractPolicyPk),
                release_date:
this.contractConfig.contractPolicy.release_date,
                allowed_timelogging_platform:
JSON.stringify(jobOffer.platform == 0 ? [1, 2] : [jobOffer.platform]),
                document_template_name: this.selectedContractName
            };

            if (jobOffer.jobDescription) {
                data.job_description_id = jobOffer.jobDescription.pk;
            }

            return this.createJobOffer(data)
                .then(response => {
                    if (response && response.pk) {
                        this.$router.push({
                            name: 'recruitment-job-offer',
                            params: {
                                ...this.$route.params,
                                job_offer_id: response.pk
                            }
                        });
                    } else {
                        let msg = `Sorry, error encountered while
creating Job offer.
                        Kindly try again.`;

                        this.errorMessage = response.error ?
response.error : msg;
                    }
                }).catch(e => {
                    console.error('Error in createJobOffer', e);
                }).finally(() => {
                    this.isSubmitting = false;
                    return data;
                });
        },

        getContractPolicyPk(params) {
            let contractPolicyPk;
            this.createContractPolicy(params)
                .then((response) => {
                    contractPolicyPk = response.pk;
                })
                .catch((e) => {
                    this.errorMessage = 'Sorry, there was an error
creating a Contract Policy';
                    console.error(e);
                });
            return contractPolicyPk;
        },

        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;
            }
        },

        async getSalaryValue(jobOffer) {
            let value = 0;
            value = parseFloat(jobOffer.compensationValue) ;

            return value.toFixed(6);
        }
    }
```

```
   };
</script>

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

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

       .header-wrapper {
           margin-bottom: 0;
           padding-top: 40px;
           padding-bottom: 24px;
           background: $white;
           top: 49px;
           z-index: 10;
       }

       .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;
           }
       }
```

```
        .body {
            margin-bottom: 52px;
            margin-top: 16px;

            .add-template {
                text-align: center;
            }
        }

        @media screen and (max-width: 1400px) {
            width: 100%;
        }
    }
</style>
```