```
<template>
  <div>
    <BaseAlert
      v-if="alertMessage"
      v-mousedown-outside="clearAlertMessage"
      :class="contractSavedSuccessfully || contractEditedSuccessfully ? 'success' : 'error'"
      icon="alert-circle-outline"
      :background="contractSavedSuccessfully || contractEditedSuccessfully ? 'success' :
'red'"
    >
      {{ alertMessage }}
    </BaseAlert>
    <BaseCard class="config">
      <aside>
        <div class="group">
          <div class="group-header">
            <p class="header-title c-secondary">
              Career Policy
            </p>
          </div>
          <div class="group-body">
            <div class="field-container">
              <div class="policy">
                <p
                  v-if="!employmentPolicy"
                  class="bold c-error"
                >
                  No career policy selected.
                </p>
                <p
                  v-else
                  class="bold c-primary"
                >
                  {{ employmentPolicy.name }}
                </p>
                <a
                  v-if="!hasEmploymentPolicy && !isAddContract"
                  @click="openEmploymentPolicyModal = true"
                >
                  {{ employmentPolicy ? 'Change' : 'Select' }}
                </a>
              </div>
            </div>
          </div>
```

```
</div>
<div class="group">
  <div class="group-header">
    <p class="header-title bold c-secondary">
      Employment
    </p>
    <p
      v-if="!isAddContract"
      class="header-subtitle t-sm c-tertiary"
    >
      Please fill in the career details correctly.
      This will help the applicant understand the nature
      of work better and will increase the chances of the
      employee coming to work for you.
    </p>
  </div>
  <div class="group-body">
    <div class="field-container">
      <p
        class="c-secondary"
        :class="!isAddContract ? 'required-field' : ''"
      >
        Name of Applicant
      </p>
      <input
        v-if="!isAddContract"
        v-model.trim="fullName"
        :class="isAddContract ? 'static-display' : ''"
        class="block applicant-name"
        placeholder="Name"
        type="text"
        disabled
      />
      <p
        v-else
        class="block static-display"
      >
        {{ fullName }}
      </p>
    </div>
    <div class="field-container">
      <p
        class="c-secondary"
        :class="!isAddContract ? 'required-field' : ''
```

```
                    >
                        Job Position
                    </p>
                    <input
                        v-if="!isAddContract"
                        v-model.trim="jobOffer.jobTitle"
                        :disabled="viewMode || isAddContract"
                        class="block job-position"
                        type="text"
                        placeholder="e.g. Web Developer"
                    />
                    <p
                        v-else
                        class="block static-display"
                    >
                        {{ jobOffer.jobTitle }}
                    </p>
                    <div v-if="hasJobDescription">
                        <div
                            v-if="!jobOffer.jobDescription"
                            class="job-desc"
                            @click="showAddJobDescriptionModal = true"
                        >
                            <i class="mdi mdi-plus-circle c-accent"></i>
                            <p class="bold c-accent">
                                Add job description
                            </p>
                        </div>
                        <div
                            v-else
                            class="job-desc"
                        >
                            <i class="mdi mdi-file-outline c-tertiary"></i>
                            <p
                                class="bold c-secondary"
                                @click="descriptionHandler"
                            >
                                {{ jobOffer.jobDescription.name }}
                            </p>
                            <div v-mousedown-outside="closeDescDropdown"
class="desc-dropdown">
                                <div
                                    class="desc-btn"
                                    :class="{open: isDescDropdownOpen}"
```

```
                @click="toggleDescDropdown"
              >
                <i class="mdi mdi-dots-vertical"></i>
              </div>
              <DropdownMenu v-if="isDescDropdownOpen">
                <ul>
                  <li
                    class="c-primary"
                    @click="viewDescriptionHandler()"
                  >
                    View
                  </li>
                  <li
                    class="c-primary"
                    @click="changeDescriptionHandler()"
                  >
                    Change
                  </li>
                  <li
                    class="c-error"
                    @click="removeDescriptionHandler()"
                  >
                    Remove
                  </li>
                </ul>
              </DropdownMenu>
            </div>
          </div>
        </div>
      </div>
      <div class="field-container">
        <p
          class="c-secondary"
          :class="!isAddContract ? 'required-field' : ''"
        >
          Job Role
        </p>
        <DropdownButton
          v-if="!isAddContract"
          :label="employmentRule.role"
          :disabled="viewMode"
          display="block"
        >
          <DropdownMenu>
```

```
            <ul>
              <li
                v-for="(role, index) in roleChoices"
                :key="`role-${index}`"
                @click="employmentRule.role = role"
              >
                {{ role }}
              </li>
            </ul>
          </DropdownMenu>
        </DropdownButton>
        <p
          v-else
          class="block static-display"
        >
          {{ employmentRule.role }}
        </p>
      </div>
      <div class="field-container">
        <p
          class="c-secondary"
          :class="!isAddContract ? 'required-field' : ''"
        >
          Partner Type
        </p>
        <DropdownButton
          v-if="!isAddContract"
          :label="employmentRule.type"
          :disabled="viewMode"
          display="block"
        >
          <DropdownMenu>
            <ul>
              <li @click="employmentRule.type = 'Exempt'">
                <p class="c-primary">
                  Exempt
                </p>
                <p class="t-sm c-tertiary">
                  Not entitled to Premium benefits
                  (Premium rates for OT, ND, and Holiday)
                </p>
              </li>
              <li @click="employmentRule.type = 'Non-exempt'">
                <p class="c-primary">
```

```
                    Non-exempt
                  </p>
                  <p class="t-sm c-tertiary">
                    Entitled to Premium benefits
                    (Premium rates for OT, ND, and Holiday)
                  </p>
                </li>
              </ul>
            </DropdownMenu>
          </DropdownButton>
          <p
            v-else
            class="block static-display"
          >
            {{ employmentRule.type }}
          </p>
        </div>
        <div class="field-container">
          <p
            class="c-secondary"
            :class="!isAddContract ? 'required-field' : ''"
          >
            Work Schedule
          </p>
          <div
            v-if="!jobOffer.schedule.name"
            class="work-sched job-desc"
            @click="clickSelectSchedule"
          >
            <i class="mdi mdi-plus-circle c-accent"></i>
            <p class="bold c-accent">
              Select Schedule
            </p>
          </div>
          <div
            v-else
            class="work-sched has-value"
            :class="!isAddContract ? 'job-desc' : ''"
          >
            <p
              :class="isAddContract ? 'static-display' : 'bold c-secondary'"
            >
              {{ jobOffer.schedule.name }}
            </p>
```

```
        <div
          v-if="!isAddContract"
          v-mousedown-outside="closeScheduleDropdown"
          class="desc-dropdown"
        >
          <div
            :class="{open: isScheduleDropdownOpen}"
            class="desc-btn"
            @click="toggleScheduleDropdown"
          >
            <i class="mdi mdi-dots-vertical"></i>
          </div>
          <DropdownMenu v-if="isScheduleDropdownOpen">
            <ul>
              <li
                class="c-primary"
                @click="selectSchedule = true; toggleScheduleDropdown()"
              >
                Change
              </li>
              <li
                class="c-error"
                @click="removeSchedule()"
              >
                Remove
              </li>
            </ul>
          </DropdownMenu>
        </div>
      </div>

      <SelectWorkScheduleModal
        v-if="selectSchedule"
        @close="selectSchedule=false"
        @schedule-group-selected="addSchedule"
      />
    </div>
    <div class="field-container">
      <p class="c-secondary">
        Location
      </p>
      <p
        :class="isAddContract ? 'static-display' : 'header-subtitle tip-light'"
      >
```

```
      {{ location }}
    </p>
  </div>
  <div class="field-container">
    <p
      class="c-secondary"
      :class="!isAddContract ? 'required-field' : ''"
    >
      Platform Access
    </p>
    <DropdownButton
      v-if="!isAddContract"
      :label="platforms[jobOffer.platform] || 'Select ApplyBPO application access'"
      :disabled="viewMode"
      display="block"
    >
      <DropdownMenu>
        <ul>
          <li
            v-for="(platform, index) in platforms"
            :key="`${platform}-${index}`"
            @click="jobOffer.platform = index"
          >
            {{ platform }}
          </li>
        </ul>
      </DropdownMenu>
    </DropdownButton>
    <p
      v-else
      class="block static-display"
    >
      {{ platforms[jobOffer.platform] }}
    </p>
  </div>
  <div class="field-container">
    <p
      class="c-secondary"
      :class="!isAddContract ? 'required-field' : ''"
    >
      Start Date
    </p>
    <DatePicker
      v-if="!isAddContract"
```

```
            :set-date="jobOffer.startDate"
            :invalid="isInvalidDate"
            :min="minDate"
            size="medium"
            @date="getDate"
          />
          <p
            v-else
            class="block static-display"
          >
            {{ startDateFormat }}
          </p>
        </div>
      </div>
    </div>
    <div
      v-if="!isAddContract"
      class="group"
    >
      <div class="group-header">
        <p class="header-title bold c-secondary">
          Compensation
        </p>
        <p class="header-subtitle t-sm c-tertiary">
          Please fill in the compensation fields correctly.
          This will dictate how the employee will be compensated in
          the course of their employment period.
        </p>
      </div>
      <div class="group-body">
        <div class="field-container">
          <ul class="tip">
            <li>
              <p> Applicant's Asking Salary </p>
              <p class="bold">
                <BaseCurrency
                  :value="applicant.salary"
                  :reference-date="applicant.salary_when"
                />
              </p>
            </li>
          </ul>
        </div>
        <div class="field-container">
```

```
<div class="field-two-grid">
  <p class="c-secondary text-container">
    Currency
  </p>
  <DropdownButton
    v-if="!isAddContract"
    :label="jobOffer.currency"
    :disabled="viewMode"
  >
    <DropdownMenu :override-close-dropdown="keepMenuOpen">
      <div class="dropdown-container">
        <div class="search">
          <input
            v-if="search"
            v-model="searchCurrency"
            type="text"
            class="block"
            placeholder="Search"
            @focus="keepMenuOpen = true"
          >
        </div>
        <ul v-if="filteredOptions.length">
          <li
            v-for="(option, index) in filteredOptions"
            :key="index"
            @click="chooseCurrency(option)"
          >
            {{ option }}
          </li>
        </ul>
        <ul v-else>
          <li>
            <p class="country">
              No currency found.
            </p>
          </li>
        </ul>
      </div>
    </DropdownMenu>
  </DropdownButton>
  <p
    v-else
    class="block static-display"
  >
```

```
      {{ jobOffer.currency }}
    </p>
  </div>
  <div class="field-two-grid">
    <p class="c-secondary text-container">
      Payable type
      <span class="tooltip-container">
        <i class="mdi mdi-help-circle-outline" />
        <Tooltip class="bottom center">
          <p class="t-sm">
            This dictates how the employee will be paid.
            For salary, the employee is paid a fixed amount
            per payroll period. For wage, the employee is
            paid per hour.
          </p>
        </Tooltip>
      </span>
    </p>
    <DropdownButton
      v-if="!isAddContract"
      :label="employmentRule.payableType"
      :disabled="viewMode"
    >
      <DropdownMenu>
        <ul>
          <li @click="selectPayableType('Wage')">
            Wage
          </li>
          <li @click="selectPayableType('Salary')">
            Salary
          </li>
        </ul>
      </DropdownMenu>
    </DropdownButton>
    <p
      v-else
      class="block static-display"
    >
      {{ employmentRule.payableType }}
    </p>
  </div>
  <div class="field-two-grid">
    <p
      class="c-secondary text-container"
```

```
                :class="!isAddContract ? 'required-field' : ""
              >
                {{ compensationLabel }}
              </p>
              <input
                v-if="!isAddContract"
                v-model="jobOffer.compensationValue"
                v-limit-character-size.number="11"
                :disabled="viewMode"
                :placeholder="employmentRule.payableType"
                class="number"
                type="number"
              />
              <p
                v-else
                class="block static-display"
              >
                {{ jobOffer.compensationValue }}
              </p>
            </div>
          </div>
        </div>
      </div>
      <div class="group">
        <div class="group-header">
          <p class="header-title bold c-secondary">
            Overtime
          </p>
        </div>
        <div class="group-body over-time">
          <div class="field-container">
            <div class="field-two-grid">
              <p class="c-secondary text-container">
                Allow Overtime
              </p>
              <DropdownButton
                v-if="!isAddContract"
                :label="employmentRule.allowOvertime"
                :disabled="viewMode"
              >
                <DropdownMenu>
                  <ul>
                    <li @click="employmentRule.allowOvertime = 'No'">
                      No
```

```
            </li>
            <li @click="employmentRule.allowOvertime = 'Yes'">
              Yes
            </li>
          </ul>
        </DropdownMenu>
      </DropdownButton>
      <p
        v-else
        class="block static-display"
      >
        {{ employmentRule.allowOvertime }}
      </p>
    </div>
    <div v-if="employmentRule.allowOvertime === 'Yes'" class="field-two-grid">
      <p class="c-secondary text-container">
        After Shift OT Request
        <span class="tooltip-container">
          <i class="mdi mdi-help-circle-outline" />
          <Tooltip class="bottom center">
            <p class="t-sm">
              Toggle whether the employee can file overtime after shift or not.
            </p>
          </Tooltip>
        </span>
      </p>
      <DropdownButton
        v-if="!isAddContract"
        :label="employmentRule.allowAfterShiftOvertime"
        :disabled="viewMode"
      >
        <DropdownMenu>
          <ul>
            <li @click="employmentRule.allowAfterShiftOvertime = 'No'">
              No
            </li>
            <li @click="employmentRule.allowAfterShiftOvertime = 'Yes'">
              Yes
            </li>
          </ul>
        </DropdownMenu>
      </DropdownButton>
      <p
        v-else
```

```
                class="block static-display"
              >
                {{ employmentRule.allowAfterShiftOvertime }}
              </p>
            </div>
            <div v-if="employmentRule.allowOvertime === 'Yes'" class="field-two-grid">
              <p class="c-secondary text-container">
                Overtime Rate (%)
              </p>
              <input
                v-if="!isAddContract"
                v-model.number="employmentRule.overtimeRate"
                v-strip-non-numeric-characters
                v-limit-character-size.number="3"
                :disabled="viewMode"
                type="number"
                placeholder="Rate"
              />
              <p
                v-else
                class="block static-display"
              >
                {{ employmentRule.overtimeRate }}
              </p>
            </div>
          </div>
        </div>
      </div>
      <div v-if="selectedTemplate && otherFields.length" class="group">
        <div class="group-header">
          <p class="header-title bold c-secondary">
            Other Fields
          </p>
        </div>
        <div class="group-body">
          <div
            v-for="(field, index) in otherFields"
            :key="`other-field-${index}`"
            class="field-container"
          >
            <div class="field-two-grid">
              <p class="c-secondary text-container required-field">
                {{ field.insert.input.placeholder }}
              </p>
```

```
                <input
                  v-if="field.insert.input.name.includes('breach-of-contract-amount')"
                  v-model="otherFieldsData.breachAmount"
                  v-limit-character-size.number="11"
                  :disabled="viewMode"
                  :placeholder="field.insert.input.placeholder"
                  :class="{ invalid: errorFields[field.insert.input.placeholder]}"
                  class="number"
                  type="number"
                  @input="handleBreachAmount()
                        errorFields[field.insert.input.placeholder] = '';"
                />
                <input
                  v-else-if="field.insert.input.name.includes('project-name')"
                  v-model="otherFieldsData.projectName"
                  :placeholder="field.insert.input.placeholder"
                  :disabled="viewMode"
                  :class="{ invalid: errorFields[field.insert.input.placeholder]}"
                  type="text"
                  @input="mirrorField(field.insert.input.name, $event.target.value)
                        errorFields[field.insert.input.placeholder] = '';"
                />
                <input
                  v-else
                  v-model="otherFieldsData.genericFields[field.insert.input.placeholder]"
                  :placeholder="field.insert.input.placeholder"
                  :disabled="viewMode"
                  :class="{ invalid: errorFields[field.insert.input.placeholder]}"
                  type="text"
                  @input="mirrorField(field.insert.input.name, $event.target.value)
                        errorFields[field.insert.input.placeholder] = '';"
                />
                <p
                  v-show="errorFields[field.insert.input.placeholder]"
                  class="c-error"
                >
                  {{ errorFields[field.insert.input.placeholder] }}
                </p>
              </div>
            </div>
          </div>
        </div>
      </aside>
      <main>
```

```
<ContractOptions
    v-if="openContractOptions"
    v-model="contractPolicy"
    :is-for-release="isForRelease"
    :start-date="jobOffer.startDate"
    :is-add-contract="isAddContract"
    :selected-template="selectedTemplate"
    :name="name"
    :has-existing-contract="hasExistingContract"
    @select-contract-template="openTemplateMenuHandler"
    @close="openContractOptions = false"
/>
<div
    v-show="(selectedTemplate || !modelValue.pk) && !openContractOptions"
    class="state-container"
>
    <div class="header">
        <div class="title">
            <p class="p-h2 c-secondary">
                {{ selectedTemplate ? selectedTemplate.name : '' }}
            </p>
            <div v-if="isEditable" class="card__footer">
                <!-- buttons for when creating new employment contract -->
                <span v-if="!modelValue.contractPk">
                    <Button
                        class="highlight error compact bold"
                        @click="$emit('delete-contract')"
                    >
                        Delete Contract
                    </Button>
                    <Button
                        v-if="!viewMode"
                        class="highlight secondary compact bold"
                        @click="openContractOptions = true"
                    >
                        Contract options
                    </Button>
                    <Button
                        class="highlight primary compact bold"
                        @click="saveContractHandler"
                    >
                        <template v-if="isSavingContract">
                            <LoadingSpinner
                                class="small white"
```

```
          />
        </template>
        <template v-else>
          Send Contract
        </template>
    </Button>
</span>

<!-- buttons for when editing/updating contract -->
<span v-else-if="isEditing">
    <Button
        class="highlight tertiary compact bold"
        @click="cancelEdit"
    >
        Cancel
    </Button>
    <Button
        v-if="!viewMode"
        class="highlight secondary compact bold"
        @click="openContractOptions = true"
    >
        Contract options
    </Button>
    <Button
        class="highlight primary compact bold"
        @click="saveContractHandler"
    >
        Send Contract
    </Button>
</span>

<!-- buttons for when updating employment contract for release -->
<span v-else-if="isForRelease">
    <Button
        class="highlight secondary compact bold"
        @click="editContractHandler"
    >
        Edit Contract
    </Button>
    <Button
        class="highlight error compact bold"
        @click="$emit('delete-contract')"
    >
        Cancel Contract
```

```
            </Button>
            <Button
              class="highlight primary compact bold"
              @click="saveContractHandler"
            >
              Send Contract
            </Button>
          </span>

          <!-- initial buttons when viewing contract -->
          <span v-else>
            <Button
              class="highlight error compact bold"
              @click="$emit('delete-contract')"
            >
              Delete Contract
            </Button>
            <Button
              class="highlight secondary compact bold"
              @click="editContractHandler"
            >
              Edit Contract
            </Button>
          </span>
        </div>
        <div v-else class="card__footer">
          <Button
            v-if="!viewMode"
            class="highlight secondary compact bold"
            @click="openContractOptions = true"
          >
            Contract options
          </Button>
        </div>
      </div>
      <EditorToolbar
        ref="toolbar"
        v-model="editor"
      />
      <input
        ref="headerPicker"
        type="file"
        class="hidden"
        @change="handleImageUpload"
```

```
            />
            <button
              v-if="!hasHeader"
              class="header-button"
              @click="openHeaderPicker"
            >
              <i class="mdi mdi-page-layout-header"></i>
              <p class="bold">
                For Business Letterhead - Add Header Image
              </p>
            </button>
          </div>
          <div class="body">
            <Editor
              ref="mirrorTarget"
              v-model="editorContent"
              :disabled="viewMode"
              @created="initializeEditor"
            />
          </div>
        </div>
      </main>
    </BaseCard>
    <SelectEmploymentPolicyModal
      v-if="openEmploymentPolicyModal"
      @close="openEmploymentPolicyModal = false"
      @select-policy="changePolicy"
    />
    <SelectContractTemplateModal
      v-if="isContractTemplateOpen"
      :employment-policy-pk="employmentPolicy.pk"
      :error-message="errorMessage"
      @select-template="selectTemplateHandler"
      @close="isContractTemplateOpen = false"
    />
    <SelectJobDescriptionModal
      v-if="showAddJobDescriptionModal"
      @add-description="addDescription"
      @close-job-desc="showAddJobDescriptionModal = false"
    />
    <JobDescriptionViewerModal
      v-if="descriptionViewer"
      :description-pk="jobOffer.jobDescription.pk"
      @close-viewer="descriptionViewer = false"
```

```
      />
    </div>
</template>

<script>
    import { mapActions, mapState } from 'vuex';
    import { shallowRef } from 'vue';

    // components
    import Editor from 'source/components/editor/Editor.vue';
    import BaseCard from 'source/components/_generics/BaseCard.vue';
    import BaseAlert from 'source/components/_generics/BaseAlert.vue';
    import Button from 'source/components/_generics/Button.vue';
    import Tooltip from 'source/components/_generics/Tooltip.vue';
    import BaseCurrency from 'source/components/_generics/BaseCurrency.vue';
    import DropdownButton from 'source/components/_generics/DropdownButton.vue';
    import DropdownMenu from 'source/components/_generics/DropdownMenu.vue';
    import SelectContractTemplateModal from
'source/components/employers/applicants/SelectContractTemplateModal.vue';
    import SelectJobDescriptionModal from
'source/components/employers/applicants/SelectJobDescriptionModal.vue';
    import JobDescriptionViewerModal from
'source/components/employers/applicants/JobDescriptionViewerModal.vue';
    import EditorToolbar from 'source/components/editor/EditorToolbar.vue';
    import ContractOptions from 'source/pages/employers/contracts/ContractOptions.vue';
    import SelectEmploymentPolicyModal from
'source/components/employers/applicants/SelectEmploymentPolicyModal.vue';
    import DatePicker from 'source/components/_generics/DatePicker.vue';
    import LoadingSpinner from 'source/components/_generics/LoadingSpinner.vue';
    import SelectWorkScheduleModal from
'source/components/employers/job-postings/popups/SelectWorkScheduleModal.vue';

    // mixins
    import TimezoneMixin from 'source/mixins/TimezoneMixin.js';
    import CurrencyMixin from 'source/mixins/CurrencyMixin.js';
    import EditorMirrorFieldMixin from 'source/mixins/EditorMirrorFieldMixin.js';
    import FieldValidatorMixin from 'source/mixins/FieldValidatorMixin.js';

    // directives
    import { StripNonNumericCharacters } from 'source/directives/StripNonNumericCharacters.js';
    import { LimitCharacterSize } from 'source/directives/LimitCharacterSize.js';
    import { MousedownOutside } from 'source/directives/MousedownOutside.js';

    export default {
```

```
name: 'ContractConfig',

components: {
    BaseCard,
    BaseAlert,
    Button,
    Tooltip,
    BaseCurrency,
    DropdownButton,
    DropdownMenu,
    SelectContractTemplateModal,
    SelectJobDescriptionModal,
    JobDescriptionViewerModal,
    EditorToolbar,
    Editor,
    ContractOptions,
    SelectEmploymentPolicyModal,
    DatePicker,
    LoadingSpinner,
    SelectWorkScheduleModal
},

directives: {
    StripNonNumericCharacters,
    LimitCharacterSize,
    MousedownOutside
},

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

inject: [
    'getAddContract',
    'getContractId'
],

props: {
    modelValue: { // for v-model
        type: Object,
        required: false
```

```
},

applicant: {
    type: Object,
    required: false
},

startDate: {
    type: String,
    required: false
},

isEditable: {
    type: Boolean,
    default: false
},

// TODO Investigate if editing/adding of schedules is still needed here
// jobScheduleGroups: {
//     type: Array,
//     required: false
// },

isForRelease: {
    type: Boolean,
    default: false
},

isSavingContract: {
    type: Boolean,
    required: false,
    default: () => false,
},

contractSaved: {
    type: Boolean,
    required: false,
    default: () => false
},
contractEdited: {
    type: Boolean,
    required: false,
    default: () => false
},
```

```
    },

    emits: ['delete-contract', 'update:modelValue', 'send-contract-name',
'clear-contract-error-message',
        'save-contract'],

    data() {
      return {
        selectSchedule: false,
        keepMenuOpen: false,
        frozenData: null,
        employmentRule: {
          role: 'Rank and File',
          type: 'Non-exempt',
          payableType: 'Wage',
          allowOvertime: 'No',
          allowAfterShiftOvertime: 'No',
          overtimeRate: 0
        },

        jobOffer: {
          jobTitle: '',
          startDate: '',
          schedule: {
            pk: -1,
            name: ''
          },
          currency: 'USD',
          compensationValue: '',
          contractContent: null,
          jobDescription: null,
          platform: -1
        },

        platforms: ['Both Mobile and Desktop', 'Desktop Only', 'Mobile Only'],
        roleChoices: ['Rank and File', 'Supervisor', 'Manager'],
        errorMessage: { status: false, message: 'Template is already used' },
        errorFields: {},

        contractPolicy: {},
        contractPk: null,
        name: 'Employment Contract',

        editorContent: {},
```

```
        editor: null,

        selectedTemplate: null,
        isContractTemplateOpen: false,
        isEmploymentPolicySelected: false,
        // createSchedule: false,
        showAddJobDescriptionModal: false,
        descriptionViewer: false,
        openContractOptions: true,

        isInitializing: true,
        isDescDropdownOpen: false,
        isEditing: false,
        alertMessage: '',
        isScheduleDropdownOpen: false,

        employmentPolicy: null,
        openEmploymentPolicyModal: false,

        otherFieldsData: { genericFields: {} },
        otherFields: [],
        isInvalidDate: true,
        searchCurrency: '',
        search: true,
        location: '',
        contractSavedSuccessfully: false,
        contractEditedSuccessfully: false,
    };
},

computed: {
    ...mapState('generics', ['currencies']),

    isAddContract() {
        return this.getAddContract();
    },

    startDateFormat() {
        return this.timezoneFormat(this.startDate, 'MMMM DD, YYYY');
    },

    viewMode() {
        return !this.isEditing;
    },
```

```
        hasEmploymentPolicy() {
            return !!this.employmentPolicy?.pk && !!this.modelValue.contractPk;
        },

        hasJobDescription() {
            return this.isAddContract === true && this.jobOffer.jobDescription;
        },

        currentDate() {
            return this.getMomentInstance().add(1, 'days').format('YYYY-MM-DD');
        },

        maxDate() {
            return this.getMomentInstance().add(10, 'years').format('YYYY-MM-DD');
        },

        employmentContractEndDate() {
            return this.getMomentInstance(this.jobOffer.startDate).add(180,
'days').format('YYYY-MM-DD');
        },

        currentRegion() {
            return this.getTimezoneGuessedRegion();
        },

        fullName() {
            return `${this.applicant.first_name} ${this.applicant.last_name}`;
        },

        compensationLabel() {
            let type = this.employmentRule.payableType === 'Wage'
                ? 'Hourly Wage'
                : 'Monthly Salary';

            return `${type} (${this.jobOffer.currency})`;
        },

        registeredFieldsWithValues() {
            return {
                'applicant-name': this.fullName,
                'applicant-address': this.applicant.address,
                'applicant-civil-status': this.applicant.civil_status_display,
                'applicant-citizenship': this.applicant.citizenship,
```

```
                'job-position': this.jobOffer.jobTitle || '',
                'job-role': this.employmentRule.role,
                'contract-start-date': this.timezoneFormat(this.jobOffer.startDate, 'MMMM DD,
YYYY'),
                'contract-end-date': this.timezoneFormat(this.employmentContractEndDate,
'MMMM DD, YYYY'),
                'training-period': 180,
                'compensation-type': this.employmentRule.payableType,
                'compensation-value': `${this.formatValue(this.jobOffer.compensationValue)}`,
            };
        },

        hasHeader() {
            if (this.editorContent) {
                let imageDeltas = this.editorContent.ops.filter(content => {
                    return (Object.keys(content.insert)).includes('image');
                });

                return imageDeltas.length > 0;
            }
            return false;
        },

        contractData: {
            get() {
                return {
                    employmentRule: this.employmentRule,
                    contractPolicy: this.contractPolicy,
                    jobOffer: this.jobOffer,
                    employmentPolicy: this.employmentPolicy,
                };
            },
            set(value) {
                this.employmentRule = value.employmentRule;
                this.contractPolicy = value.contractPolicy;
                this.jobOffer = value.jobOffer;
                this.employmentPolicy = value.employmentPolicy;
            }
        },

        filteredOptions() {
            return this.currencies.filter(opt =>
opt.toLowerCase().includes(this.searchCurrency.toLowerCase()));
        },
```

```
    hasExistingContract() {
      return Boolean(this.modelValue.pk) || Boolean(this.modelValue.contractPk);
    }
  },

  watch: {
    contractData: {
      handler() {
        this.$emit('update:modelValue', {
          ...this.contractData,
          contractPk: this.contractPk,
          name: this.selectedTemplate ? this.selectedTemplate.name : this.name
        });
      },
      deep: true
    },

    jobOffer: {
      handler(jobOffer) {
        localStorage.setItem('jobOffer', JSON.stringify(this.jobOffer));
      },
      deep: true,
    },

    employmentPolicy: {
      handler(employmentPolicy) {
        localStorage.setItem('employmentPolicy', JSON.stringify(this.employmentPolicy));
      },
      deep: true,
    },

    employmentRule: {
      handler(employmentRule) {
        localStorage.setItem('employmentRule', JSON.stringify(this.employmentRule));
      },
      deep: true,
    },

    editorContent: {
      handler(content) {
        this.jobOffer.contractContent = content;

        this.editorMirrorConfig = {
```

```
            transformAllInputToText: false,
            content: content
        };
    },
    deep: true
},

otherFieldsData: {
    handler() {
        let amount = this.otherFieldsData.breachAmount || 0;
        this.mirrorField('breach-of-contract-amount', `${this.formatValue(amount)}`);
    },
    deep: true
},

'jobOffer.currency': {
    handler() {
        let amount = this.otherFieldsData.breachAmount || 0;
        this.mirrorField('breach-of-contract-amount', `${this.formatValue(amount)}`);
    },
    deep: true
},

selectedTemplate: {
    handler() {
        if (this.selectedTemplate) {
            let inputDeltas = this.editorContent?.ops?.filter(content => {
                return (Object.keys(content.insert)).includes('input');
            });

            let registeredFields = Object.keys(this.registeredFieldsWithValues);
            let newlyRegistered = [];
            this.otherFields = [];
            this.errorFields = {};

            inputDeltas.forEach(content => {
                let name = content.insert.input.name;
                let placeholder = content.insert.input.placeholder;
                let fieldIsRegistered = registeredFields.includes(name);

                if (!fieldIsRegistered && !newlyRegistered.includes(name)) {
                    newlyRegistered.push(name);
                    this.otherFields.push(content);
                    if (name.includes('breach-of-contract-amount')) {
```

```
                        this.otherFieldsData.breachAmount = '';
                    } else if (name.includes('project-name')) {
                        this.otherFieldsData.projectName = '';
                    } else {
                        this.otherFieldsData.genericFields[placeholder] = '';
                    }
                }
            });

        } else {
            this.otherFields = null;
        }
    },
    deep: true
},

contractSavedSuccessfully() {
    if (this.contractSavedSuccessfully) {
        this.alertMessage = 'Contract has been sent Successfully';
    } else {
        this.alertMessage = '';
    }
},

contractEditedSuccessfully() {
    if (this.contractEditedSuccessfully) {
        this.alertMessage = 'Updated contract has been sent successfully';
    } else {
        this.alertMessage = '';
    }
},

contractSaved() {
    this.contractSavedSuccessfully = this.contractSaved;
},

contractEdited() {
    this.contractEditedSuccessfully = this.contractEdited;
},
},

mounted() {
    let cachedjobOffer = JSON.parse(localStorage.getItem('jobOffer'));
    if (cachedjobOffer)
```

```
            this.jobOffer = cachedjobOffer;

        let cachedEmploymentPolicy = JSON.parse(localStorage.getItem('employmentPolicy'));
        if (cachedEmploymentPolicy)
            this.employmentPolicy = cachedEmploymentPolicy;

        let catchedEmploymentRule = JSON.parse(localStorage.getItem('employmentRule'));
        if (catchedEmploymentRule)
            this.employmentRule = catchedEmploymentRule;

    },

    async created() {
        this.jobOffer.currency = this.currency;
        this.jobOffer.platform = this.applicant?.job_offers[0].allowed_timelogging_platform[0];
        this.jobOffer.jobDescription = this.applicant?.job_offers[0].job_description;

        const modelValue = JSON.parse(JSON.stringify(this.modelValue));
        if (modelValue.pk || this.isAddContract === true) {
            this.contractPk = modelValue.pk ? modelValue.pk : modelValue.contractPk;
            const filter = this.contractPk ? this.contractPk : parseInt(this.getContractId()) ;
            this.getEmploymentContract(filter)
                .then(async (response) => {
                    let employmentRule = response.employment_rule;

                    this.employmentPolicy = response.employment_policy;
                    this.contractPolicy = response.contract_policy || {};
                    this.contractPolicy.release_date = this.timezoneFormat(response.release_date,
'YYYY-MM-DD');

                    if (this.contractPk) {
                        this.name = response.name;
                        this.contractPolicy.release_date =
this.timezoneFormat(response.release_date, 'YYYY-MM-DD');
                    } else {
                        this.contractPolicy.release_date = this.timezoneFormat(null, 'YYYY-MM-DD');
                    }


                    this.employmentRule.role = employmentRule.position_display;
                    this.employmentRule.pk = employmentRule.pk;
                    this.employmentRule.type = employmentRule.employee_type_display;
                    this.employmentRule.overtimeRate = employmentRule.overtime_rate * 100;
```

```
                this.employmentRule.allowOvertime = employmentRule.can_overtime ? 'Yes' :
'No',
                this.employmentRule.payableType =
                    employmentRule.compensation_type_display === 'Hourly' ? 'Wage' : 'Salary',
                this.employmentRule.allowAfterShiftOvertime =
                    employmentRule.can_file_overtime_after_shift? 'Yes' : 'No';

                let compensation = await this.convertCurrency(
                    this.currency, this.jobOffer.currency, response.salary_rate.amount);
                compensation = this.employmentRule.payableType === 'Salary' ?
                    compensation * 12 : compensation;
                this.jobOffer.compensationValue = compensation.toFixed(2).toString();
                this.jobOffer.contractContent = response.content;
                this.jobOffer.jobTitle = response.job_position;
                this.jobOffer.startDate = this.timezoneFormat(this.currentDate, 'YYYY-MM-DD');
                this.jobOffer.schedule = response.schedule;
                this.location = response.office_location ? response.office_location.full_address :
'';

                if (modelValue.pk || modelValue.contractPk) {
                    const parsedContent = typeof response.content === 'object' ?
                        response.content : JSON.parse(response.content);
                    this.$nextTick(() => {
                        this.editorContent = parsedContent;
                        this.editorMirrorConfig = {
                            transformAllInputToText: false,
                            content: this.editorContent
                        };
                        this.mirrorFields();
                    });
                    this.openContractOptions = false;
                }
            })
            .catch(e => {
                console.error('Error in getEmploymentContract', e);
            });
        }

        this.frozenData = JSON.parse(JSON.stringify(this.contractData));
        this.jobPostingInfo();
        this.isInitializing = false;
        this.contractSavedSuccessfully = this.contractSaved;
    },
```

```
methods: {
    ...mapActions('template', ['getTemplate']),
    ...mapActions('contract', ['getEmploymentContract']),
    ...mapActions('employers', [
        'fetchJobPostingById',
        'fetchJobScheduleGroups'
    ]),

    async jobPostingInfo() {
        let jobPostingPk = this.$route.params.job_posting_pk;
        if (jobPostingPk) {
            let response = await this.fetchJobPostingById(jobPostingPk);
            this.location = !response.error ? response.job_location : '';
        }
    },

    changePolicy(policy) {
        this.employmentPolicy = policy;
        this.openEmploymentPolicyModal = false;
    },

    descriptionHandler() {
        this.descriptionViewer = true;
    },

    viewDescriptionHandler() {

        this.descriptionViewer = true;
        this.toggleDescDropdown();

    },

    changeDescriptionHandler() {
        this.showAddJobDescriptionModal = true;
        this.toggleDescDropdown();
    },

    removeDescriptionHandler() {
        this.jobOffer.jobDescription = null;
        this.toggleDescDropdown();
    },

    addDescription(description) {
        this.jobOffer.jobDescription = description;
```

```
    this.showAddJobDescriptionModal = false;
},

cancelEdit() {
    this.contractData = JSON.parse(JSON.stringify(this.frozenData));
    this.isEditing = false;
},

clearAlertMessage() {
    this.alertMessage = '';
},

formatValue(value) {
    let formatter = new Intl.NumberFormat(undefined, {
        minimumFractionDigits: 2
    });
    return `${this.jobOffer.currency} ${formatter.format(Number(value))}`;
},

selectPayableType(type) {
    this.jobOffer.compensationValue = '';
    this.employmentRule.payableType = type;
},

formatDate(date) {
    return this.timezoneFormat(date, 'MMMM DD, YYYY');
},

async selectTemplateHandler(template) {
    this.selectedTemplate = await this.getTemplate(template.pk);
    this.isContractTemplateOpen = false;
    this.editorContent = JSON.parse(this.selectedTemplate.content);
    this.editorMirrorConfig = {
        transformAllInputToText: false,
        content: this.editorContent
    };
    this.isEditing = true;
    this.openContractOptions = false;
    this.mirrorFields();
    this.$emit('send-contract-name', this.selectedTemplate.name);
    this.$emit('clear-contract-error-message');
},

openHeaderPicker() {
```

```
    this.$refs.headerPicker.click();
},

handleImageUpload(event) {
    this.$refs.toolbar.handleImageUpload(event);
},

toggleDescDropdown() {
    this.isDescDropdownOpen = !this.isDescDropdownOpen;
},

toggleScheduleDropdown() {
    this.isScheduleDropdownOpen = !this.isScheduleDropdownOpen;
},

closeDescDropdown() {
    this.isDescDropdownOpen = false;
},

closeScheduleDropdown() {
    this.isScheduleDropdownOpen = false;
},

removeSchedule() {
    this.jobOffer.schedule.pk = -1;
    this.jobOffer.schedule.name = '';
    this.isScheduleDropdownOpen = !this.isScheduleDropdownOpen;
},

saveContractHandler() {
    if (this.hasEmptyField()) {
        return;
    }

    try {
        this.validateQuillDocumentNotEmpty(this.jobOffer.contractContent);
        this.validateNotEmpty('Job Position', this.jobOffer.jobTitle);
        this.validateNotNegative(this.jobOffer.schedule.pk);
        const dataToValidate = {
            'Start Date': this.jobOffer.startDate,
            'Compensation Value': this.jobOffer.compensationValue,
        };
        const hasProjectName = this.otherFields?.filter((field) => {
            return field.insert.input.name.includes('project-name');
```

```
        });
        if (hasProjectName?.length) {
            dataToValidate['Project Name'] = this.otherFieldsData.projectName;
        }
        this.validateNoEmptyFields(dataToValidate);
    } catch (err) {
        if (err === 'Negative Value') {
            this.alertMessage = 'Please select a schedule';
        } else if (err === 'Empty Document') {
            this.alertMessage = 'Please add a contract';
        } else if (err.error === 'empty') {
            this.alertMessage = `${err.key} must not be empty`;
        }
        return;
    }

    this.alertMessage = '';
    this.$emit('save-contract');
    this.frozenData = JSON.parse(JSON.stringify(this.contractData));
    if (this.contractPk) {
        this.isEditing = false;
    }
},

openTemplateMenuHandler() {
    this.modelValue.employmentPolicy
        ? this.isContractTemplateOpen = true
        : this.alertMessage = 'Please select a career policy first.';
},

getDate(date) {
    if (date <= this.maxDate && date >= this.currentDate) {
        this.isInvalidDate = false;
        this.jobOffer.startDate = date;
    } else {
        this.isInvalidDate = true;
    }
},

chooseCurrency(curr) {
    this.keepMenuOpen = false;
    this.jobOffer.currency = curr;
},
```

```
        addSchedule(schedule) {
            this.jobOffer.schedule = schedule;
        },

        clickSelectSchedule() {
            if (!this.isInitializing || !this.viewMode)
            {
                this.selectSchedule=true;
            }
            else
            {
                this.selectSchedule=false;
            }
        },

        hasEmptyField() {
            this.errorFields = {};

            if (this.otherFieldsData?.breachAmount?.trim() === '') {
                this.errorFields['Breach of Contract Amount'] = 'Breach of Contract Amount is
required';
            }

            if (this.otherFieldsData?.projectName?.trim() === '') {
                this.errorFields['Project Name'] = 'Project Name is required';
            }

            for (const fieldName in this.otherFieldsData.genericFields) {
                if (this.otherFieldsData.genericFields?.[fieldName]?.trim() === '') {
                    this.errorFields[fieldName] = `${fieldName} is required`;
                }
            }
            return Object.keys(this.errorFields).length > 0;
        },

        handleBreachAmount() {
            let amount = this.otherFieldsData.breachAmount || 0;
            this.mirrorField('breach-of-contract-amount', `${this.formatValue(amount)}`);
        },

        initializeEditor(editorInstance) {
            this.editor = shallowRef(editorInstance);
        },
```

```
        editContractHandler() {
            this.frozenData = JSON.parse(JSON.stringify(this.contractData));
            this.isEditing = true;
        }
    },
};
</script>

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

    .card {
        height: 880px;
        min-height: 110%;
        display: grid;
        grid-template-columns: auto 1fr;
        margin-bottom: 24px;

        aside {
            width: 360px;
            background: $primary-highlight;
            border-right: 1px solid $quaternary;
            overflow-y: auto;
            @include scrollbar-thin();

            .group {
                padding: 24px 16px;

                &:not(:last-child) {
                    box-shadow: 0 1px 0 0 $quaternary;
                }

                &-header {
                    margin-bottom: 20px;

                    .header-title {
                        margin-bottom: 4px;
                    }

                    .header-subtitle {
                        &:not(:last-child) {
                            margin-bottom: 16px;
                        }
                    }
```

```
}

&-body {
    .field-container {
        &:not(:last-child) {
            margin-bottom: 16px;
        }

        input,
        .dropdown {
            margin: 6px 0;
        }

        .policy {
            display: flex;

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

            a {
                cursor: pointer;
            }
        }

        .work-sched {
            margin: 6px 0;

            &.has-value {
                &:hover {
                    cursor: default;
                    background: $quaternary !important;
                }
            }

            .desc-dropdown {

                .desc-btn {
                    cursor: pointer;
                    text-align: center;
                }
```

```
        &:hover {
            background: darken($quaternary, 4%);
        }
    }
}

.dropdown-menu {
    display: block;
    width: 100%;
    max-height: 200px;
    overflow-y: auto;
    @include scrollbar-thin();

    ul {
        @include clear-list-formatting();
        padding: 8px 0;

        li {
            padding: 8px 12px;
            user-select: none;
            cursor: pointer;

            &:hover {
                background: lighten($quaternary, 5%);
            }
        }

        hr {
            margin: 8px 0;
            border: 0;
            border-bottom: 1px solid $quaternary;
        }
    }

    a {
        cursor: pointer;
        text-decoration: none;
        padding: 8px 12px;
    }
}

.schedule {
    &:not(:last-child) {
        border-bottom: 1px solid $quaternary;
```

```
    }

    &-name {
        margin-bottom: 8px;
    }

    &-day {
        margin-top: 6px;

        .day {
            margin-bottom: 4px;
        }
    }

}

.workforce-item {
    display: flex;

    img {
        border-radius: $corner-radius;
        height: 40px;
        width: 56px;
        object-fit: cover;
    }

    .information {
        flex: 1;
        padding-left: 12px;
        align-self: center;
    }
}

.field-two-grid {
    display: flex;
    flex-direction: column;

    .text-container {
        align-self: flex-start;
        margin-bottom: 5px;
    }

    .tooltip-container {
        vertical-align: middle;
```

```
        i {
            font-size: 18px;
        }

        .tooltip {
            width: 200px;
        }
    }

    ::v-deep .dropdown button,
    input {
        align-self: flex-start;
        min-width: 0;
        width: 100%;
        padding: 0 12px;
    }
}

.tip {
    padding: 10px 12px;
    border-radius: $corner-radius;
    background: $quaternary;
}

.tip-light {
    padding: 10px 12px;
    border-radius: $corner-radius;
    background: $white;
}

ul.tip {
    @include clear-list-formatting();
    margin-top: 6px;
    padding: 10px 12px;

    li {
        display: flex;
        color: $secondary;

        &:not(:last-child) {
            margin-bottom: 8px;
        }
```

```
        p {
            font-size: 12px;

            &:first-child {
                flex: 1;
                padding-right: 8px;
            }
        }
    }
}

.salary-container {
    display: flex;

    :deep(.dropdown button ) {
        min-width: 96px;
        border-radius: 4px 0 0 4px;
    }

    input {
        flex: 1;
        border-radius: 0 4px 4px 0;
        border-left: none;
    }
}

.compensation-list {
    .compensation {
        display: flex;

        &:not(:last-child) {
            margin-bottom: 8px;
        }

        p {
            flex: 1;
            align-self: center;
            padding-right: 12px;
        }

        input {
            width: 120px;
        }
```

```
        i {
            font-size: 16px;
            color: $tertiary;
        }

        .tooltip-container {
            vertical-align: middle;
        }

        .tooltip {
            width: 160px;
        }

        .dropdown,
        input {
            margin: 0;
        }

        :deep(.dropdown button ) {
            min-width: 120px;
            width: 120px;
            padding: 0 12px;
        }
    }
}

.job-desc {
    display: flex;
    align-items: center;
    padding: 0 12px;
    height: 40px;
    border-radius: $corner-radius;
    background: $quaternary;
    cursor: pointer;
    user-select: none;

    i {
        font-size: 20px;
    }

    p {
        flex: 1;
        @include clamp-line(1);
        padding: 0 8px;
```

```
}

.desc-dropdown {
    display: inline-block;
    position: relative;
    border-radius: $corner-radius;
    margin-right: -8px;

    .desc-btn {
        display: flex;
        align-items: center;
        padding: 4px;

        i {
            color: $tertiary;
        }

        &.open {
            background: $accent-highlight;

            i {
                color: $accent;
            }
        }
    }

    .dropdown-menu {
        top: 0;
        left: -130px;
        width: 160px;

        ul {
            @include clear-list-formatting();
            margin: 8px 0;

            li {
                padding: 8px 16px;

                &:hover {
                    background: $primary-highlight;
                }
            }
        }
    }
```

```
                }


                &:hover {
                    background: darken($quaternary, 4%);
                }
            }
        }
    }
}

&__footer {
    padding: 10px;
    justify-content: space-between;

    span {
        display: flex;
        align-items: center;
    }
}

main {
    position: relative;
    display: grid;

    :deep(.ql-container ) {
        overflow-y: auto;

        .ql-editor {
            @include scrollbar-thin();
            max-height: 780px;
        }
    }

    .state-container {

        .header-button {
            position: absolute;
            display: flex;
            z-index: 1;
            width: 100%;
            flex-direction: column;
            align-items: center;
```

```
    border: none;
    background: $accent-highlight;
    outline: none;
    cursor: pointer;
    user-select: none;
    border-radius: $corner-radius;
    color: $accent;
    padding: 14px;
    transition-duration: .2s;

    i {
        font-size: 32px;
    }

    p {
        margin-top: 8px;
    }

    &:hover {
        background: #B3DDF3;
    }

    &[disabled] {
        background: #EFF8FC;
        color: #2e3133;
    }
}

.header {
    background: $white;
    position: sticky;
    top: 0;
    z-index: 2;

    .title {
        height: 56px;
        display: flex;
        align-items: center;
        padding: 0 8px 0 16px;
        box-shadow: 0 1px 0 0 $quaternary;

        p {
            flex: 1;
        }
```

```
                }
            }

            .body {
                margin-bottom: 28px 0px;
            }
        }
    }
}

.alert :deep( ) {
    margin-bottom: 10px;
}

.error {
    color: $error;
}
.static-display {
    padding: 0px;
    margin-top: 8px;
}
</style>
```