```
<template>
  <div class="details">
    <div class="details__header">
      <div class="information">
        <Badge
          v-if="hasEmptyFields"
          size="small"
        />
        <p class="t-md semibold">
          Additional Fields
        </p>
      </div>
      <div class="icon-container c-tertiary">
        <i class="mdi mdi-list-box-outline" />
        <p>
          {{ localApplicantContractFields.length }}
          {{ localApplicantContractFields.length > 1 ? 'fields' : 'field' }}
        </p>
      </div>
      <DisclosureButton
        alternate
        :toggled="isOpen"
        @toggle-state="isOpen = !isOpen"
      />
    </div>
    <ul
      v-show="isOpen"
      class="details__body"
    >
      <li
        v-for="(field, index) in localApplicantContractFields"
        :key="`other-field-${index}`"
        class="field-item"
      >
        <div class="label">
          <p class="required-field">
            {{ getAddressLabel(field) }}
          </p>
        </div>
        <div class="value">
          <template v-if="field.name.includes('address')">
            <div>
              <DropdownButton
                :label="countryLabel ? countryLabel : 'Select Country'"
                class="block"
                :class="{'invalid': isFieldEmpty(addressParts.country)
                  && !!fieldErrors['applicant-address']}"
                :disabled="isUpdatingContract || !hasApplicantFillableFields"
              >
                <DropdownMenu
                  class="block"
                  search-bar
                  :override-close-dropdown="!filteredCountries.length"
                  placeholder="Search country"
                  @search="countrySearchKey = $event"
                >
                  <ul>
                    <template v-if="filteredCountries.length">
                      <li
                        v-for="country in filteredCountries"
                        :key="`${country.id}-${country.pk}`"
                        @click="updateAddressPart('country', country)"
                      >
                        {{ country.name }}
                      </li>
                    </template>
                    <li v-else class="not-selectable">
                      <p class="c-tertiary">
                        No results found
                      </p>
                    </li>
                  </ul>
                </DropdownMenu>
              </DropdownButton>
            </div>
            <div>
              <DropdownButton
                :label="provinceLabel ? provinceLabel : 'Select Province'"
                class="block"
                :class="{'invalid': isFieldEmpty(addressParts.province)
                  && !!fieldErrors['applicant-address']}"
                :disabled="!countryLabel || !hasProvinces
                  || isUpdatingContract || !hasApplicantFillableFields"
              >
                <DropdownMenu
                  class="block"
                  search-bar
                  :override-close-dropdown="!filteredProvinces.length"
                  placeholder="Search province"
                  @search="provinceSearchKey = $event"
                >
                  <ul>
                    <template v-if="filteredProvinces.length">
                      <li
                        v-for="province in filteredProvinces"
                        :key="`${province.id}-${province.pk}`"
                        @click="updateAddressPart('province', province)"
                      >
                        {{ province.name }}
                      </li>
                    </template>
                    <li v-else class="not-selectable">
                      <p class="c-tertiary">
                        No results found
                      </p>
                    </li>
                  </ul>
                </DropdownMenu>
              </DropdownButton>
            </div>
            <div>
              <input
                id="street-address"
                v-model="streetLabel"
                class="block"
                type="text"
                placeholder="Enter Street Address"
                :disabled="isUpdatingContract || !hasApplicantFillableFields"
                @input="updateAddressPart('street', $event.target.value)"
              />
            </div>
          </template>
          <template v-else-if="field.name.includes('civil-status')">
            <DropdownButton
              :label="field.value || 'Select civil status'"
              class="block"
              :class="{'invalid':isFieldEmpty(field.value)
                && !!fieldErrors[field.name]}"
              :disabled="isUpdatingContract || !hasApplicantFillableFields"
            >
              <DropdownMenu class="block">
                <ul>
```

```
                <li
                    v-for="status in civilStatuses"
                    :key="status.id + status.value"
                    :value="status.value"
                    @click="updateFieldValue(field, status.value)"
                >
                    {{ status.value }}
                </li>
            </ul>
        </DropdownMenu>
    </DropdownButton>
    <p
        v-if="!field.value"
        class="t-sm c-error"
    >
        {{ fieldErrors[field.name] }}
    </p>
</template>
<template v-else>
    <input
        v-model="field.value"
        class="block"
        type="text"
        :placeholder="setPlaceholder(field.placeholder)"
        :class="{'invalid':isFieldEmpty(field.value)
            && !!fieldErrors[field.name]}"
        :disabled="isUpdatingContract || !hasApplicantFillableFields"
        @input="updateFieldValue(field, $event.target.value)"
    />
    <p
        v-if="!field.value"
        class="t-sm c-error"
    >
        {{ fieldErrors[field.name] }}
    </p>
</template>
</div>
</li>
<Button
    v-if="localApplicantContractFields.length && hasApplicantFillableFields"
    :class="`primary
        ${windowWidth <= MOBILE ? 'block small' : ''}
    `"
    :disabled="isUpdatingContract || isUpdateButtonDisabled"
```

```
            @click="updateApplicantContractFields"
        >
            Update
        </Button>
      </ul>
   </div>
</template>

<script>
   import { mapGetters, mapActions } from 'vuex';
   import Button from 'source/components/_generics/Button.vue';
   import DisclosureButton from 'source/components/_generics/DisclosureButton.vue';
   import DropdownButton from 'source/components/_generics/DropdownButton.vue';
   import DropdownMenu from 'source/components/_generics/DropdownMenu.vue';
   import CompensationConversionMixin from 'source/mixins/CompensationConversionMixin.js';
   import CurrencyMixin from 'source/mixins/CurrencyMixin.js';
   import EditorMirrorFieldMixin from 'source/mixins/EditorMirrorFieldMixin.js';
   import Badge from 'source/components/_generics/Badge.vue';


   export default {
      name: 'ContractFields',

      components: {
         DisclosureButton,
         Button,
         DropdownButton,
         DropdownMenu,
         Badge
      },

      mixins: [
         EditorMirrorFieldMixin,
         CurrencyMixin,
         CompensationConversionMixin
      ],


      props: {
         selectedContract: {
            type: Object,
            required: true
         }
      },
      emits: ['update-field'],

      data() {
         return {
            isOpen: true,
            windowWidth: window.innerWidth,
            MOBILE: 576,
            fillableFields: [
                'applicant-civil-status',
                'applicant-citizenship'
            ],
            fieldErrors: {
                'applicant-civil-status': '',
                'applicant-address': '',
                'applicant-citizenship': ''
            },
            fieldErrorMessages: {
                'applicant-civil-status': 'Please select applicant civil status.',
                'applicant-address': 'Please enter applicant address.',
                'applicant-citizenship': 'Please enter applicant citizenship.'
            },
            civilStatuses: [
                { id: 1, value: 'Single' },
                { id: 2, value: 'Married' },
                { id: 3, value: 'Separated' },
                { id: 4, value: 'Divorced' },
                { id: 5, value: 'Widowed' }
            ],
            civilStatus: '',
            localApplicantContractFields: [],
            countrySearchKey: '',
            provinceSearchKey: '',
            province: this.getStoredProvince(),
            country: this.getStoredCountry(),
            provinces: [],
            location: {
                country: '',
                province: '',
            },
            isFetchingProvinces: false,
            countryLabel: '',
            provinceLabel: '',
            streetLabel: '',
            isUpdatingContract: false
        };
    },

    computed: {
        ...mapGetters('generics', ['countries']),

        hasProvinces() {
            return this.provinces.length !== 0;
        },

        filteredCountries() {
            if (this.countrySearchKey.length) {
                return this.countries.filter(country => country.code.toLowerCase()
                    .match(this.countrySearchKey.toLowerCase()) || country.name.toLowerCase()
                        .match(this.countrySearchKey.toLowerCase()));
            } else {
                return this.countries;
            }
        },

        filteredProvinces() {
            if (this.provinceSearchKey.length) {
                return this.provinces.filter(province => province.name.toLowerCase()
                    .match(this.provinceSearchKey.toLowerCase()));
            } else {
                return this.provinces;
            }
        },

        addressParts() {
            const address = this.localApplicantContractFields.find(
                field => field.name === 'applicant-address');
            if (address) {
                const parts = address.value.split(', ');
                return {
                    street: parts[0],
                    province: parts[1],
                    country: parts[2]
                };
            } else {
                return {
                    street: '',
                    province: '',
                    country: ''
                };
            }
        },

        isUpdateButtonDisabled() {
            return this.countryLabel === '' || this.streetLabel === '';
        },

        hasApplicantFillableFields() {
            return this.selectedContract?.has_applicant_fillable_fields;
        },

        hasEmptyFields() {
            return this.localApplicantContractFields.some(field => !field.value);
        },

        hasFieldErrors() {
            return Object.values(this.fieldErrors).some(value => value !== '');
        }
    },

    watch: {
        'selectedContract.applicant_fillable_fields': {
            handler(value) {
                this.localApplicantContractFields = value || [];
            }
        },
        country: {
            async handler(newCountry) {
                if (!newCountry || !newCountry.pk) {
                    this.province = '';
                    this.provinceLabel = '';
                    this.provinces = [];
                    return;
                }

                this.isFetchingProvinces = true;
                await this.fetchProvincesHandler();
                this.isFetchingProvinces = false;

                if (this.provinces.length === 0) {
                    this.province = '';
                    this.provinceLabel = '';
                }
            },
            immediate: true
        }
    },

    mounted() {
        window.addEventListener('resize', this.handleResize);
        this.handleResize();
    },

    beforeUnmount() {
        window.removeEventListener('resize', this.handleResize);
    },

    created() {
        // Deep clone to avoid reactivity issue
        this.localApplicantContractFields = JSON.parse(
            JSON.stringify(this.selectedContract.applicant_fillable_fields));
        this.isOpen = this.hasApplicantFillableFields;
        this.fetchCountriesHandler();

        const addressField = this.localApplicantContractFields.find(
            field => field.name === 'applicant-address');
        if (addressField) {
            if (addressField.value) {
                this.updateFieldValue(addressField, addressField.value);
            } else {
                this.country = JSON.parse(localStorage.getItem('selectedCountry')) || {};
                this.countryLabel = this.country.name || '';
                const province = JSON.parse(localStorage.getItem('selectedProvince')) || {};
                this.province = { name: province.name, pk: province.pk };
                this.provinceLabel = province.name || '';
                this.streetLabel = (localStorage.getItem('selectedStreet')) || '';
            }
        }
    },

    methods: {
        ...mapActions('employers', ['fetchProvinces']),
        ...mapActions('generics', ['fetchCountries']),
        ...mapActions('jobOffer', ['updateApplicantContract']),
        async fetchCountriesHandler() {
            await this.fetchCountries();
        },

        async fetchProvincesHandler() {
            this.isFetchingProvinces = true;
            this.provinces = await this.fetchProvinces(this.country.pk);
            this.isFetchingProvinces = false;
        },

        async updateApplicantContractFields() {
            this.setFieldErrors();

            if (this.hasFieldErrors) return;

            this.isUpdatingContract = true;
            let contractContent = this.selectedContract.content;

            if (!contractContent?.ops) {
                contractContent = JSON.parse(contractContent);
            }

            const inputs = contractContent.ops.filter(content =>
                content.insert && Object.prototype.hasOwnProperty.call(content.insert, 'input')
            );

            // Use dictionary for faster lookups
            const fieldMap = Object.fromEntries(
                this.localApplicantContractFields.map(field => [field.name, field.value]));

            // Update applicant field values of contractContent
            inputs.forEach(content => {
                const inputName = content.insert.input.name;
                if (fieldMap.hasOwnProperty(inputName)) {
                    content.insert.input.value = fieldMap[inputName];
                }
            });

            try {
                const data = {
                    'pk': this.selectedContract.pk,
                    'content': JSON.stringify(contractContent),
                    'action': 'content_update'
                };
                const response = await this.updateApplicantContract(data);
                this.$eventBus.$emit('employment-activated');
                this.$eventBus.$emit('contract-updated', response);
            } catch {
                this.clearFieldErrors();
            } finally {
                this.isUpdatingContract = false;
            }
        },

        updateLocation() {
            let validCountry = this.countries.find((el) => {
                return el.pk == this.country.pk;
            });

            let validProvince = this.provinces.find((el) => {
                return el.pk == this.province.pk;
            });

            this.addressParts.country = validCountry ? validCountry.name : '';
            this.addressParts.province = validProvince ? validProvince.name : '';
            this.addressParts.street = '';
        },

        handleResize() {
            this.windowWidth = window.innerWidth;
        },

        isFieldEmpty(value) {
            return value === '';
        },

        updateFieldValue(field, value) {
            field.value = value;
            this.$emit('update-field', {
                name: field.name,
                value
            });

            if (field.name == 'applicant-civil-status') {
                this.civilStatus = value;
            }
```

```
        if (field.name === 'applicant-address') {
            const addressParts = value.split(', ');
            if (addressParts.length === 3) {
                this.countryLabel = addressParts[2];
                this.provinceLabel = addressParts[1];
                this.streetLabel = addressParts[0];
            }
        }
    },

    updateAddressPart(part, value) {
        const addressField = this.localApplicantContractFields.find(
            field => field.name === 'applicant-address');
        if (addressField) {
            if (part === 'street') {
                this.streetLabel = value;
                localStorage.setItem('selectedStreet', JSON.stringify(value));
            } else if (part === 'province') {
                this.province = value;
                this.provinceLabel = value.name;
                localStorage.setItem('selectedProvince', JSON.stringify({ name: value.name, pk:
value.pk }));
            } else if (part === 'country') {
                this.country = value;
                this.countryLabel = value.name;
                localStorage.setItem('selectedCountry', JSON.stringify(value));
            }

            const addressParts = [this.streetLabel, this.provinceLabel, this.countryLabel]
                .filter(part => part && part.trim() !== '')
                .join(', ');

            this.updateFieldValue(addressField, addressParts);
        }
    },

    setPlaceholder(placeholder) {
        return placeholder || 'Enter value';
    },

    getAddressLabel(field) {
        if (field.name.includes('address')) {
            const labels = ['Country', 'Province', 'Street Address'];
            return labels.join(',  \n');
```

```
                }
                return field.placeholder;
            },

            setFieldErrors() {
                this.clearFieldErrors();

                this.localApplicantContractFields.forEach(field => {
                    const { name: fieldName, value: fieldValue } = field;

                    if (this.fillableFields.includes(fieldName) && !fieldValue) {
                        this.fieldErrors[fieldName] = this.fieldErrorMessages[fieldName];
                    } else {
                        this.fieldErrors[fieldName] = '';
                    }
                });
            },

            clearFieldErrors() {
                this.fieldErrors = {
                    'applicant-civil-status': '',
                    'applicant-citizenship': ''
                };
            },

            getStoredProvince() {
                try {
                    return JSON.parse(localStorage.getItem('selectedProvince')) ?? '';
                } catch {
                    return '';
                }
            },

            getStoredCountry() {
                try {
                    return JSON.parse(localStorage.getItem('selectedCountry')) ?? '';
                } catch {
                    return '';
                }
            }
        },
    };
</script>
```

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

  .details {
    border: 1px solid $border;
    border-radius: 8px;
    background: var(--sem-card);

    &__header {
      @include flex-align-center;
      gap: 16px;
      padding: 12px 16px;
      min-height: 56px;

      .information {
        @include flex-align-center;
        gap: 8px;
        flex: 1;
      }

      .icon-container {
        @include inline-flex-align-center;
        gap: 8px ;

        i {
          font-size: 18px;
        }
      }

      @media #{$mobile} {
        padding: 8px 12px;
        min-height: 0;
        gap: 12px;
      }
    }

    &__body {
      @include clear-list-formatting;
      @include flex-column;
      gap: 12px;
      padding: 12px 16px;
      border-top: 1px solid $border;

      .field-item {
```

```
    display: flex;
    gap: 12px;

    .label {
        @include flex-align-center;
        height: 40px;
        width: 240px;

        p {
            color: $tertiary;
        }
    }

    .value {
        @include flex-column;
        gap: 4px;
        flex: 1;
    }
}

button {
    margin-left: auto;
}

@media #{$mobile} {
    padding: 12px;

    .field-item {
        flex-direction: column;
        align-items: flex-start;
        gap: 4px;

        .label {
            widows: 100%;

            p {
                font-size: 12px;
            }
        }

        .value {
            width: 100%;
        }
    }
```

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

```
<template>
   <div class="details">
       <div class="details__header">
           <div class="information">
               <Badge
                   v-if="hasEmptyFields"
                   size="small"
              />
               <p class="t-md semibold">
                   Additional Fields
               </p>
           </div>
           <div class="icon-container c-tertiary">
               <i class="mdi mdi-list-box-outline" />
               <p>
                   {{ localApplicantContractFields.length }}
                   {{ localApplicantContractFields.length > 1 ? 'fields' :
'field' }}
               </p>
           </div>
           <DisclosureButton
               alternate
               :toggled="isOpen"
               @toggle-state="isOpen = !isOpen"
          />
       </div>
       <ul
           v-show="isOpen"
           class="details__body"
       >
           <li
               v-for="(field, index) in localApplicantContractFields"
               :key="`other-field-${index}`"
               class="field-item"
           >
               <div class="label">
```

```
                   <p class="required-field">
                       {{ getAddressLabel(field) }}
                   </p>
               </div>
               <div class="value">
                   <template v-if="field.name.includes('address')">
                       <div>
                           <DropdownButton
                               :label="countryLabel ? countryLabel :
'Select Country'"
                               class="block"
                               :class="{'invalid':
isFieldEmpty(addressParts.country)
                                   && !!fieldErrors['applicant-address']}"
                               :disabled="isUpdatingContract ||
!hasApplicantFillableFields"
                           >
                               <DropdownMenu
                                   class="block"
                                   search-bar

:override-close-dropdown="!filteredCountries.length"
                                   placeholder="Search country"
                                   @search="countrySearchKey = $event"
                               >
                                   <ul>
                                       <template
v-if="filteredCountries.length">
                                           <li
                                               v-for="country in
filteredCountries"

:key="`${country.id}-${country.pk}`"

@click="updateAddressPart('country', country)"
                                           >
                                               {{ country.name }}
                                           </li>
                                       </template>
                                       <li v-else class="not-selectable">
```

```
                                            <p class="c-tertiary">
                                                No results found
                                            </p>
                                        </li>
                                    </ul>
                                </DropdownMenu>
                            </DropdownButton>
                        </div>
                        <div>
                            <DropdownButton
                                :label="provinceLabel ? provinceLabel :
'Select Province'"
                                class="block"
                                :class="{'invalid':
isFieldEmpty(addressParts.province)
                                    && !!fieldErrors['applicant-address']}"
                                :disabled="!countryLabel || !hasProvinces
                                    || isUpdatingContract ||
!hasApplicantFillableFields"
                            >
                                <DropdownMenu
                                    class="block"
                                    search-bar

:override-close-dropdown="!filteredProvinces.length"
                                    placeholder="Search province"
                                    @search="provinceSearchKey = $event"
                                >
                                    <ul>
                                        <template
v-if="filteredProvinces.length">
                                            <li
                                                v-for="province in
filteredProvinces"

:key="`${province.id}-${province.pk}`"

@click="updateAddressPart('province', province)"
                                            >
                                                {{ province.name }}
```

```
                                           </li>
                                       </template>
                                       <li v-else class="not-selectable">
                                           <p class="c-tertiary">
                                               No results found
                                           </p>
                                       </li>
                                   </ul>
                               </DropdownMenu>
                           </DropdownButton>
                       </div>
                       <div>
                           <input
                               id="street-address"
                               v-model="streetLabel"
                               class="block"
                               type="text"
                               placeholder="Enter Street Address"
                               :disabled="isUpdatingContract ||
!hasApplicantFillableFields"
                               @input="updateAddressPart('street',
$event.target.value)"
                           />
                       </div>
                   </template>
                   <template
v-else-if="field.name.includes('civil-status')">
                       <DropdownButton
                           :label="field.value || 'Select civil status'"
                           class="block"
                           :class="{'invalid':isFieldEmpty(field.value)
                               && !!fieldErrors[field.name]}"
                           :disabled="isUpdatingContract ||
!hasApplicantFillableFields"
                       >
                           <DropdownMenu class="block">
                               <ul>
                                   <li
                                       v-for="status in civilStatuses"
                                       :key="status.id + status.value"
                                       :value="status.value"
                                       @click="updateFieldValue(field,
status.value)"
                                   >
                                       {{ status.value }}
                                   </li>
                               </ul>
                           </DropdownMenu>
                       </DropdownButton>
                       <p
                           v-if="!field.value"
                           class="t-sm c-error"
                       >
                           {{ fieldErrors[field.name] }}
                       </p>
                   </template>
                   <template v-else>
                       <input
                           v-model="field.value"
                           class="block"
                           type="text"

:placeholder="setPlaceholder(field.placeholder)"
                           :class="{'invalid':isFieldEmpty(field.value)
                               && !!fieldErrors[field.name]}"
                           :disabled="isUpdatingContract ||
!hasApplicantFillableFields"
                           @input="updateFieldValue(field,
$event.target.value)"
                       />
                       <p
                           v-if="!field.value"
                           class="t-sm c-error"
                       >
                           {{ fieldErrors[field.name] }}
                       </p>
                   </template>
               </div>
           </li>
           <Button
```

```
               v-if="localApplicantContractFields.length &&
hasApplicantFillableFields"
               :class="`primary
                   ${windowWidth <= MOBILE ? 'block small' : ''}
               `"
               :disabled="isUpdatingContract || isUpdateButtonDisabled"
               @click="updateApplicantContractFields"
           >
               Update
           </Button>
       </ul>
   </div>
</template>

<script>
   import { mapGetters, mapActions } from 'vuex';
   import Button from 'source/components/_generics/Button.vue';
   import DisclosureButton from
'source/components/_generics/DisclosureButton.vue';
   import DropdownButton from
'source/components/_generics/DropdownButton.vue';
   import DropdownMenu from
'source/components/_generics/DropdownMenu.vue';
   import CompensationConversionMixin from
'source/mixins/CompensationConversionMixin.js';
   import CurrencyMixin from 'source/mixins/CurrencyMixin.js';
   import EditorMirrorFieldMixin from
'source/mixins/EditorMirrorFieldMixin.js';
   import Badge from 'source/components/_generics/Badge.vue';


   export default {
       name: 'ContractFields',

       components: {
           DisclosureButton,
           Button,
           DropdownButton,
           DropdownMenu,
           Badge
```

```
        },

        mixins: [
            EditorMirrorFieldMixin,
            CurrencyMixin,
            CompensationConversionMixin
        ],


        props: {
            selectedContract: {
                type: Object,
                required: true
            }
        },

        emits: ['update-field'],

        data() {
            return {
                isOpen: true,
                windowWidth: window.innerWidth,
                MOBILE: 576,
                fillableFields: [
                    'applicant-civil-status',
                    'applicant-citizenship'
                ],
                fieldErrors: {
                    'applicant-civil-status': '',
                    'applicant-address': '',
                    'applicant-citizenship': ''
                },
                fieldErrorMessages: {
                    'applicant-civil-status': 'Please select applicant
civil status.',
                    'applicant-address': 'Please enter applicant address.',
                    'applicant-citizenship': 'Please enter applicant
citizenship.'
                },
                civilStatuses: [
                    { id: 1, value: 'Single' },
                    { id: 2, value: 'Married' },
                    { id: 3, value: 'Separated' },
                    { id: 4, value: 'Divorced' },
                    { id: 5, value: 'Widowed' }
                ],
                civilStatus: '',
                localApplicantContractFields: [],
                countrySearchKey: '',
                provinceSearchKey: '',
                province: this.getStoredProvince(),
                country: this.getStoredCountry(),
                provinces: [],
                location: {
                    country: '',
                    province: '',
                },
                isFetchingProvinces: false,
                countryLabel: '',
                provinceLabel: '',
                streetLabel: '',
                isUpdatingContract: false
            };
        },

        computed: {
            ...mapGetters('generics', ['countries']),

            hasProvinces() {
                return this.provinces.length !== 0;
            },

            filteredCountries() {
                if (this.countrySearchKey.length) {
                    return this.countries.filter(country =>
country.code.toLowerCase()
                        .match(this.countrySearchKey.toLowerCase()) ||
country.name.toLowerCase()
                            .match(this.countrySearchKey.toLowerCase()));
                } else {
                    return this.countries;
                }
            },

            filteredProvinces() {
                if (this.provinceSearchKey.length) {
                    return this.provinces.filter(province =>
province.name.toLowerCase()
                        .match(this.provinceSearchKey.toLowerCase()));
                } else {
                    return this.provinces;
                }
            },

            addressParts() {
                const address = this.localApplicantContractFields.find(
                    field => field.name === 'applicant-address');
                if (address) {
                    const parts = address.value.split(', ');
                    return {
                        street: parts[0],
                        province: parts[1],
                        country: parts[2]
                    };
                } else {
                    return {
                        street: '',
                        province: '',
                        country: ''
                    };
                }
            },

            isUpdateButtonDisabled() {
                return this.countryLabel === '' || this.streetLabel === '';
            },

            hasApplicantFillableFields() {
                return
this.selectedContract?.has_applicant_fillable_fields;
```

```
            },

            hasEmptyFields() {
                return this.localApplicantContractFields.some(field =>
!field.value);
            },

            hasFieldErrors() {
                return Object.values(this.fieldErrors).some(value => value
!== '');
            }
        },

        watch: {
            'selectedContract.applicant_fillable_fields': {
                handler(value) {
                    this.localApplicantContractFields = value || [];
                }
            },
            country: {
                async handler(newCountry) {
                    if (!newCountry || !newCountry.pk) {
                        this.province = '';
                        this.provinceLabel = '';
                        this.provinces = [];
                        return;
                    }

                    this.isFetchingProvinces = true;
                    await this.fetchProvincesHandler();
                    this.isFetchingProvinces = false;

                    if (this.provinces.length === 0) {
                        this.province = '';
                        this.provinceLabel = '';
                    }
                },
                immediate: true
            }
        },
        mounted() {
            window.addEventListener('resize', this.handleResize);
            this.handleResize();
        },

        beforeUnmount() {
            window.removeEventListener('resize', this.handleResize);
        },

        created() {
            // Deep clone to avoid reactivity issue
            this.localApplicantContractFields = JSON.parse(

JSON.stringify(this.selectedContract.applicant_fillable_fields));
            this.isOpen = this.hasApplicantFillableFields;
            this.fetchCountriesHandler();

            const addressField = this.localApplicantContractFields.find(
                field => field.name === 'applicant-address');
            if (addressField) {
                if (addressField.value) {
                    this.updateFieldValue(addressField,
addressField.value);
                } else {
                    this.country =
JSON.parse(localStorage.getItem('selectedCountry')) || {};
                    this.countryLabel = this.country.name || '';
                    const province =
JSON.parse(localStorage.getItem('selectedProvince')) || {};
                    this.province = { name: province.name, pk: province.pk
};

                    this.provinceLabel = province.name || '';
                    this.streetLabel =
(localStorage.getItem('selectedStreet')) || '';
                }
            }
        },

        methods: {
            ...mapActions('employers', ['fetchProvinces']),
            ...mapActions('generics', ['fetchCountries']),
            ...mapActions('jobOffer', ['updateApplicantContract']),


            async fetchCountriesHandler() {
                await this.fetchCountries();
            },

            async fetchProvincesHandler() {
                this.isFetchingProvinces = true;
                this.provinces = await
this.fetchProvinces(this.country.pk);
                this.isFetchingProvinces = false;
            },

            async updateApplicantContractFields() {
                this.setFieldErrors();

                if (this.hasFieldErrors) return;

                this.isUpdatingContract = true;
                let contractContent = this.selectedContract.content;

                if (!contractContent?.ops) {
                    contractContent = JSON.parse(contractContent);
                }

                const inputs = contractContent.ops.filter(content =>
                    content.insert &&
Object.prototype.hasOwnProperty.call(content.insert, 'input')
                );

                // Use dictionary for faster lookups
                const fieldMap = Object.fromEntries(
                    this.localApplicantContractFields.map(field =>
[field.name, field.value]));

                // Update applicant field values of contractContent
                inputs.forEach(content => {
                    const inputName = content.insert.input.name;
                    if (fieldMap.hasOwnProperty(inputName)) {
                        content.insert.input.value = fieldMap[inputName];
                    }
                });

                try {
                    const data = {
                        'pk': this.selectedContract.pk,
                        'content': JSON.stringify(contractContent),
                        'action': 'content_update'
                    };

                    const response = await
this.updateApplicantContract(data);
                    this.$eventBus.$emit('employment-activated');
                    this.$eventBus.$emit('contract-updated', response);
                } catch {
                    this.clearFieldErrors();
                } finally {
                    this.isUpdatingContract = false;
                }
            },

            updateLocation() {
                let validCountry = this.countries.find((el) => {
                    return el.pk == this.country.pk;
                });

                let validProvince = this.provinces.find((el) => {
                    return el.pk == this.province.pk;
                });

                this.addressParts.country = validCountry ?
validCountry.name : '';
                this.addressParts.province = validProvince ?
validProvince.name : '';
                this.addressParts.street = '';
            },
            handleResize() {
                this.windowWidth = window.innerWidth;
            },

            isFieldEmpty(value) {
                return value === '';
            },

            updateFieldValue(field, value) {
                field.value = value;
                this.$emit('update-field', {
                    name: field.name,
                    value
                });

                if (field.name == 'applicant-civil-status') {
                    this.civilStatus = value;
                }

                if (field.name === 'applicant-address') {
                    const addressParts = value.split(', ');
                    if (addressParts.length === 3) {
                        this.countryLabel = addressParts[2];
                        this.provinceLabel = addressParts[1];
                        this.streetLabel = addressParts[0];
                    }
                }
            },

            updateAddressPart(part, value) {
                const addressField =
this.localApplicantContractFields.find(
                    field => field.name === 'applicant-address');
                if (addressField) {
                    if (part === 'street') {
                        this.streetLabel = value;
                        localStorage.setItem('selectedStreet',
JSON.stringify(value));
                    } else if (part === 'province') {
                        this.province = value;
```

```
                       this.provinceLabel = value.name;
                       localStorage.setItem('selectedProvince',
JSON.stringify({ name: value.name, pk: value.pk }));
                   } else if (part === 'country') {
                       this.country = value;
                       this.countryLabel = value.name;
                       localStorage.setItem('selectedCountry',
JSON.stringify(value));
                   }

                   const addressParts = [this.streetLabel,
this.provinceLabel, this.countryLabel]
                       .filter(part => part && part.trim() !== '')
                       .join(', ');

                   this.updateFieldValue(addressField, addressParts);
               }
           },

           setPlaceholder(placeholder) {
               return placeholder || 'Enter value';
           },

           getAddressLabel(field) {
               if (field.name.includes('address')) {
                   const labels = ['Country', 'Province', 'Street
Address'];
                   return labels.join(',  \n');
               }
               return field.placeholder;
           },

           setFieldErrors() {
               this.clearFieldErrors();

               this.localApplicantContractFields.forEach(field => {
                   const { name: fieldName, value: fieldValue } = field;

                   if (this.fillableFields.includes(fieldName) &&
!fieldValue) {
```

```
                        this.fieldErrors[fieldName] =
this.fieldErrorMessages[fieldName];
                    } else {
                        this.fieldErrors[fieldName] = '';
                    }
                });
            },

            clearFieldErrors() {
                this.fieldErrors = {
                    'applicant-civil-status': '',
                    'applicant-citizenship': ''
                };
            },

            getStoredProvince() {
                try {
                    return
JSON.parse(localStorage.getItem('selectedProvince')) ?? '';
                } catch {
                    return '';
                }
            },

            getStoredCountry() {
                try {
                    return
JSON.parse(localStorage.getItem('selectedCountry')) ?? '';
                } catch {
                    return '';
                }
            }
        },
    };
</script>

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

   .details {
```

```scss
    border: 1px solid $border;
    border-radius: 8px;
    background: var(--sem-card);

    &__header {
        @include flex-align-center;
        gap: 16px;
        padding: 12px 16px;
        min-height: 56px;

        .information {
            @include flex-align-center;
            gap: 8px;
            flex: 1;
        }

        .icon-container {
            @include inline-flex-align-center;
            gap: 8px ;

            i {
                font-size: 18px;
            }
        }

        @media #{$mobile} {
            padding: 8px 12px;
            min-height: 0;
            gap: 12px;
        }
    }

    &__body {
        @include clear-list-formatting;
        @include flex-column;
        gap: 12px;
        padding: 12px 16px;
        border-top: 1px solid $border;

        .field-item {
```

```scss
        display: flex;
        gap: 12px;

        .label {
            @include flex-align-center;
            height: 40px;
            width: 240px;

            p {
                color: $tertiary;
            }
        }

        .value {
            @include flex-column;
            gap: 4px;
            flex: 1;
        }
    }

    button {
        margin-left: auto;
    }

    @media #{$mobile} {
        padding: 12px;

        .field-item {
            flex-direction: column;
            align-items: flex-start;
            gap: 4px;

            .label {
                widows: 100%;

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

```
                    .value {
                        width: 100%;
                    }
                }
            }
        }
    }
</style>
```