List

https://playdocs1.orangeriver-ad055946.westus2.azurecontainerapps.io/play-docs/docs/ui-components/Data-Display/List

List

A versatile, feature-rich list component that supports single and multi-selection, integrates with Angular forms, and provides extensive customization options including avatars, icons, and action buttons. Built with accessibility in mind and designed for complex data display scenarios.

How to use

```
import { AavaListComponent, AavaListItemsComponent } from "@aava/play-core";
```

```
import { AavaListComponent, AavaListItemsComponent } from "@aava/play-core";
```

Note: The List component is standalone and includes all necessary dependencies. TheAavaListItemsComponentis used for individual list items with content projection.

```
AavaListItemsComponent
```

Basic Usage

Simple list implementations with avatars, icons, and basic selection functionality.

```
<aava-list>  <aava-list-items *ngFor="let profile of userProfiles">    <div left>      <aava-avatars        size="large"        shape="pill"        [imageUrl]="sampleImageUrl"      ></aava-avatars>    </div>    <div middle>      <h4>{{ profile.heading }}</h4>      <p>{{ profile.description }}</p>    </div>    <div right>      <aava-icon        [iconName]="'arrow-right'"        iconColor="#000000ff"        iconSize="24"      ></aava-icon>    </div>  </aava-list-items></aava-list>
```

```
<aava-list>  <aava-list-items *ngFor="let profile of userProfiles">    <div left>      <aava-avatars        size="large"        shape="pill"        [imageUrl]="sampleImageUrl"      ></aava-avatars>    </div>    <div middle>      <h4>{{ profile.heading }}</h4>      <p>{{ profile.description }}</p>    </div>    <div right>      <aava-icon        [iconName]="'arrow-right'"        iconColor="#000000ff"        iconSize="24"      ></aava-icon>    </div>  </aava-list-items></aava-list>
```

```
sampleImageUrl = "assets/1.svg";userProfiles = [  {    id: 1,    heading: "Heading comes here",    description: "Description text goes here",    avatarUrl: "https://randomuser.me/api/portraits/men/1.jpg",    iconName: "chevron-right",    button: {      text: "label",      variant: "primary" as ButtonVariant,      color: "#1976d2",      action: "view_profile",    },  },  {    id: 2,    heading: "Heading comes here",    description: "Description text goes here",    avatarUrl: "https://randomuser.me/api/portraits/women/2.jpg",    iconName: "chevron-right",    button: {      text: "label",      variant: "secondary" as ButtonVariant,      color: "#388e3c",      action: "contact",    },  },  {    id: 3,    heading: "Heading comes here",    description: "Description text goes here",    avatarUrl: "https://randomuser.me/api/portraits/men/3.jpg",    iconName: "chevron-right",    button: {      text: "label",      variant: "primary" as ButtonVariant,      color: "#f57c00",      action: "view_portfolio",    },  },];
```

```
sampleImageUrl = "assets/1.svg";userProfiles = [  {    id: 1,    heading: "Heading comes here",    description: "Description text goes here",    avatarUrl: "https://randomuser.me/api/portraits/men/1.jpg",    iconName: "chevron-right",    button: {      text: "label",      variant: "primary" as ButtonVariant,      color: "#1976d2",      action: "view_profile",    },  },  {    id: 2,    heading: "Heading comes here",    description: "Description text goes here",    avatarUrl: "https://randomuser.me/api/portraits/women/2.jpg",    iconName: "chevron-right",    button: {      text: "label",      variant: "secondary" as ButtonVariant,      color: "#388e3c",      action: "contact",    },  },  {    id: 3,    heading: "Heading comes here",    description: "Description text goes here",    avatarUrl: "https://randomuser.me/api/portraits/men/3.jpg",    iconName: "chevron-right",    button: {      text: "label",      variant: "primary" as ButtonVariant,      color: "#f57c00",      action: "view_portfolio",    },  },];
```

Multi-Selection

Advanced multi-selection capabilities with checkboxes, selection limits, and programmatic control.

```
import { Component } from '@angular/core';import { ListComponent } from '@aava/play-comp-library';@Component({  selector: 'app-list-multi-select',  standalone: true,  imports: [ListComponent],  template: `    <div class="demo-
container">      <h3>Multi-Selection List</h3>      <div class="multi-select-examples">        <div
class="example-section">          <h4>Basic Multi-Select</h4>          <aava-list            [items]="basicItems"
[title]="'Select Multiple Items'"            [multiSelect]="true"
(onSelectionChanged)="onSelectionChanged($event)"          ></aava-list>        </div>        <div class="example-
section">          <h4>Multi-Select with Checkboxes</h4>          <aava-list            [items]="checkboxItems"
[title]="'Select with Checkboxes'"            [multiSelect]="true"            [showCheckboxes]="true"
(onSelectionChanged)="onCheckboxSelectionChanged($event)"          ></aava-list>        </div>        <div
class="example-section">          <h4>Limited Multi-Select (Max 3)</h4>          <aava-list
[items]="limitedItems"            [title]="'Select Up to 3 Items'"            [multiSelect]="true"
[maxSelections]="3"            (onSelectionChanged)="onLimitedSelectionChanged($event)"          ></aava-list>
</div>      </div>      <div class="control-buttons">        <button (click)="selectAll()" class="btn btn-
primary">Select All</button>        <button (click)="clearSelection()" class="btn btn-secondary">Clear
Selection</button>        <button (click)="selectSpecific()" class="btn btn-success">Select Items 1, 3, 5</button>
</div>      <div class="selection-outputs">        <div class="output-section">          <h4>Basic Multi-Select
Output</h4>          <div class="output-content">            <p><strong>Selected Items:</strong> {{
basicSelection.selectedItems.map(item => item.title).join(', ') || 'None' }}</p>            <p><strong>Selected
IDs:</strong> {{ basicSelection.selectedIds.join(', ') || 'None' }}</p>            <p><strong>Count:</strong> {{
basicSelection.selectedItems.length }}</p>          </div>        </div>        <div class="output-section">
<h4>Checkbox Selection Output</h4>          <div class="output-content">            <p><strong>Selected Items:
</strong> {{ checkboxSelection.selectedItems.map(item => item.title).join(', ') || 'None' }}</p>            <p>
<strong>Selected IDs:</strong> {{ checkboxSelection.selectedIds.join(', ') || 'None' }}</p>            <p>
<strong>Count:</strong> {{ checkboxSelection.selectedItems.length }}</p>          </div>        </div>        <div
class="output-section">          <h4>Limited Selection Output</h4>          <div class="output-content">
<p><strong>Selected Items:</strong> {{ limitedSelection.selectedItems.map(item => item.title).join(', ') || 'None'
}}</p>            <p><strong>Selected IDs:</strong> {{ limitedSelection.selectedIds.join(', ') || 'None' }}</p>
<p><strong>Count:</strong> {{ limitedSelection.selectedItems.length }} / 3</p>            <p><strong>Can Select
More:</strong> {{ limitedSelection.selectedItems.length < 3 ? 'Yes' : 'No' }}</p>          </div>        </div>
</div>      <div class="usage-tips">        <h4>Multi-Selection Tips:</h4>        <ul>          <li><strong>Multi-
Select Mode:</strong> Set `multiSelect="true"` to enable multiple selections</li>          <li><strong>Checkboxes:
</strong> Use `showCheckboxes="true"` for visual checkbox indicators</li>          <li><strong>Max Selections:
</strong> Set `maxSelections` to limit the number of selectable items</li>          <li><strong>Event Handling:
</strong> Use `onSelectionChanged` for multi-select events</li>          <li><strong>Programmatic Control:
</strong> Use `selectAll()`, `clearSelection()`, and `selectItems()` methods</li>        </ul>      </div>
</div>  `,  styles: [`    .demo-container {      max-width: 1200px;      margin: 20px 0;    }    .multi-select-
examples {      display: grid;      grid-template-columns: repeat(auto-fit, minmax(350px, 1fr));      gap: 32px;
margin: 20px 0;    }    .example-section {      padding: 20px;      border: 1px solid #e9ecef;      border-radius:
8px;      background: #f8f9fa;    }    .example-section h4 {      margin-top: 0;      margin-bottom: 16px;
color: #495057;      font-size: 16px;    }    .control-buttons {      display: flex;      gap: 12px;      margin:
24px 0;      flex-wrap: wrap;    }    .btn {      padding: 8px 16px;      border: none;      border-radius: 4px;
cursor: pointer;      font-size: 14px;      transition: background-color 0.2s;    }    .btn-primary {
background: #007bff;      color: white;    }    .btn-primary:hover {      background: #0056b3;    }    .btn-
secondary {      background: #6c757d;      color: white;    }    .btn-secondary:hover {      background: #545b62;
}    .btn-success {      background: #28a745;      color: white;    }    .btn-success:hover {      background:
#1e7e34;    }    .selection-outputs {      display: grid;      grid-template-columns: repeat(auto-fit,
minmax(300px, 1fr));      gap: 20px;      margin: 24px 0;    }    .output-section {      padding: 16px;
border: 1px solid #e9ecef;      border-radius: 6px;      background: #f8f9fa;    }    .output-section h4 {
margin-top: 0;      margin-bottom: 12px;      color: #495057;      font-size: 14px;    }    .output-content p {
margin: 4px 0;      font-size: 13px;      color: #495057;    }    .usage-tips {      margin-top: 24px;
padding: 16px;      background: #d1f2eb;      border-radius: 6px;      border-left: 4px solid #20c997;    }
.usage-tips h4 {      margin-top: 0;      color: #0f5132;    }    .usage-tips ul {      margin: 8px 0;
padding-left: 20px;    }    .usage-tips li {      margin: 4px 0;      color: #0f5132;    }    .usage-tips strong {
color: #051b11;    }    @media (max-width: 768px) {      .multi-select-examples {        grid-template-columns:
1fr;        gap: 20px;      }      .selection-outputs {        grid-template-columns: 1fr;        gap: 16px;
}      .control-buttons {        flex-direction: column;      }    }  `]})export class ListMultiSelectDemo {
basicSelection = { selectedItems: [], selectedIds: [] };  checkboxSelection = { selectedItems: [], selectedIds: []
};  limitedSelection = { selectedItems: [], selectedIds: [] };  basicItems = [    { id: '1', title: 'Item 1',
subtitle: 'First item' },    { id: '2', title: 'Item 2', subtitle: 'Second item' },    { id: '3', title: 'Item 3',
subtitle: 'Third item' },    { id: '4', title: 'Item 4', subtitle: 'Fourth item' },    { id: '5', title: 'Item 5',
subtitle: 'Fifth item' }  ];  checkboxItems = [    { id: '1', title: 'Task 1', subtitle: 'Complete documentation'
},    { id: '2', title: 'Task 2', subtitle: 'Review code' },    { id: '3', title: 'Task 3', subtitle: 'Write
tests' },    { id: '4', title: 'Task 4', subtitle: 'Deploy to staging' },    { id: '5', title: 'Task 5', subtitle:
'Update dependencies' }  ];  limitedItems = [    { id: '1', title: 'Option A', subtitle: 'First option' },    {
id: '2', title: 'Option B', subtitle: 'Second option' },    { id: '3', title: 'Option C', subtitle: 'Third option'
},    { id: '4', title: 'Option D', subtitle: 'Fourth option' },    { id: '5', title: 'Option E', subtitle: 'Fifth
option' }  ];  onSelectionChanged(event: any) {    this.basicSelection = event;    console.log('Basic selection
changed:', event);  }  onCheckboxSelectionChanged(event: any) {    this.checkboxSelection = event;
console.log('Checkbox selection changed:', event);  }  onLimitedSelectionChanged(event: any) {
this.limitedSelection = event;    console.log('Limited selection changed:', event);  }  selectAll() {    // This
would be called on the list component reference    console.log('Select all clicked');  }  clearSelection() {    //
This would be called on the list component reference    console.log('Clear selection clicked');  }
selectSpecific() {    // This would be called on the list component reference    console.log('Select specific
items clicked');  }}
```

```
import { Component } from '@angular/core';import { ListComponent } from '@aava/play-comp-library';@Component({
selector: 'app-list-multi-select',  standalone: true,  imports: [ListComponent],  template: `    <div class="demo-
container">      <h3>Multi-Selection List</h3>      <div class="multi-select-examples">        <div
class="example-section">          <h4>Basic Multi-Select</h4>          <aava-list            [items]="basicItems"
[title]="'Select Multiple Items'"            [multiSelect]="true"
(onSelectionChanged)="onSelectionChanged($event)"          ></aava-list>        </div>        <div class="example-
section">          <h4>Multi-Select with Checkboxes</h4>          <aava-list            [items]="checkboxItems"
[title]="'Select with Checkboxes'"            [multiSelect]="true"            [showCheckboxes]="true"
(onSelectionChanged)="onCheckboxSelectionChanged($event)"          ></aava-list>        </div>        <div
class="example-section">          <h4>Limited Multi-Select (Max 3)</h4>          <aava-list
[items]="limitedItems"           [title]="'Select Up to 3 Items'"           [multiSelect]="true"
[maxSelections]="3"           (onSelectionChanged)="onLimitedSelectionChanged($event)"          ></aava-list>
</div>      </div>      <div class="control-buttons">        <button (click)="selectAll()" class="btn btn-
primary">Select All</button>        <button (click)="clearSelection()" class="btn btn-secondary">Clear
Selection</button>        <button (click)="selectSpecific()" class="btn btn-success">Select Items 1, 3, 5</button>
</div>      <div class="selection-outputs">        <div class="output-section">          <h4>Basic Multi-Select
Output</h4>          <div class="output-content">            <p><strong>Selected Items:</strong> {{
basicSelection.selectedItems.map(item => item.title).join(', ') || 'None' }}</p>            <p><strong>Selected
IDs:</strong> {{ basicSelection.selectedIds.join(', ') || 'None' }}</p>            <p><strong>Count:</strong> {{
basicSelection.selectedItems.length }}</p>          </div>        </div>        <div class="output-section">
<h4>Checkbox Selection Output</h4>          <div class="output-content">            <p><strong>Selected Items:
</strong> {{ checkboxSelection.selectedItems.map(item => item.title).join(', ') || 'None' }}</p>            <p>
<strong>Selected IDs:</strong> {{ checkboxSelection.selectedIds.join(', ') || 'None' }}</p>            <p>
<strong>Count:</strong> {{ checkboxSelection.selectedItems.length }}</p>          </div>        </div>        <div
class="output-section">          <h4>Limited Selection Output</h4>          <div class="output-content">
<p><strong>Selected Items:</strong> {{ limitedSelection.selectedItems.map(item => item.title).join(', ') || 'None'
}}</p>            <p><strong>Selected IDs:</strong> {{ limitedSelection.selectedIds.join(', ') || 'None' }}</p>
<p><strong>Count:</strong> {{ limitedSelection.selectedItems.length }} / 3</p>            <p><strong>Can Select
More:</strong> {{ limitedSelection.selectedItems.length < 3 ? 'Yes' : 'No' }}</p>          </div>        </div>
</div>      <div class="usage-tips">        <h4>Multi-Selection Tips:</h4>        <ul>          <li><strong>Multi-
Select Mode:</strong> Set `multiSelect="true"` to enable multiple selections</li>          <li><strong>Checkboxes:
</strong> Use `showCheckboxes="true"` for visual checkbox indicators</li>          <li><strong>Max Selections:
</strong> Set `maxSelections` to limit the number of selectable items</li>          <li><strong>Event Handling:
</strong> Use `onSelectionChanged` for multi-select events</li>          <li><strong>Programmatic Control:
</strong> Use `selectAll()`, `clearSelection()`, and `selectItems()` methods</li>        </ul>      </div>
</div>  `,  styles: [`    .demo-container {      max-width: 1200px;      margin: 20px 0;    }    .multi-select-
examples {      display: grid;      grid-template-columns: repeat(auto-fit, minmax(350px, 1fr));      gap: 32px;
margin: 20px 0;    }    .example-section {      padding: 20px;      border: 1px solid #e9ecef;      border-radius:
8px;      background: #f8f9fa;    }    .example-section h4 {      margin-top: 0;      margin-bottom: 16px;
color: #495057;      font-size: 16px;    }    .control-buttons {      display: flex;      gap: 12px;      margin:
24px 0;      flex-wrap: wrap;    }    .btn {      padding: 8px 16px;      border: none;      border-radius: 4px;
cursor: pointer;      font-size: 14px;      transition: background-color 0.2s;    }    .btn-primary {
background: #007bff;      color: white;    }    .btn-primary:hover {      background: #0056b3;    }    .btn-
secondary {      background: #6c757d;      color: white;    }    .btn-secondary:hover {      background: #545b62;
}    .btn-success {      background: #28a745;      color: white;    }    .btn-success:hover {      background:
#1e7e34;    }    .selection-outputs {      display: grid;      grid-template-columns: repeat(auto-fit,
minmax(300px, 1fr));      gap: 20px;      margin: 24px 0;    }    .output-section {      padding: 16px;
border: 1px solid #e9ecef;      border-radius: 6px;      background: #f8f9fa;    }    .output-section h4 {
margin-top: 0;      margin-bottom: 12px;      color: #495057;      font-size: 14px;    }    .output-content p {
margin: 4px 0;      font-size: 13px;      color: #495057;    }    .usage-tips {      margin-top: 24px;
padding: 16px;      background: #d1f2eb;      border-radius: 6px;      border-left: 4px solid #20c997;    }
.usage-tips h4 {      margin-top: 0;      color: #0f5132;    }    .usage-tips ul {      margin: 8px 0;
padding-left: 20px;    }    .usage-tips li {      margin: 4px 0;      color: #0f5132;    }    .usage-tips strong {
color: #051b11;    }    @media (max-width: 768px) {      .multi-select-examples {        grid-template-columns:
1fr;        gap: 20px;      }      .selection-outputs {        grid-template-columns: 1fr;        gap: 16px;
}      .control-buttons {        flex-direction: column;      }    }  `]})export class ListMultiSelectDemo {
basicSelection = { selectedItems: [], selectedIds: [] };  checkboxSelection = { selectedItems: [], selectedIds: []
};  limitedSelection = { selectedItems: [], selectedIds: [] };  basicItems = [    { id: '1', title: 'Item 1',
subtitle: 'First item' },    { id: '2', title: 'Item 2', subtitle: 'Second item' },    { id: '3', title: 'Item 3',
subtitle: 'Third item' },    { id: '4', title: 'Item 4', subtitle: 'Fourth item' },    { id: '5', title: 'Item 5',
subtitle: 'Fifth item' }  ];  checkboxItems = [    { id: '1', title: 'Task 1', subtitle: 'Complete documentation'
},    { id: '2', title: 'Task 2', subtitle: 'Review code' },    { id: '3', title: 'Task 3', subtitle: 'Write
tests' },    { id: '4', title: 'Task 4', subtitle: 'Deploy to staging' },    { id: '5', title: 'Task 5', subtitle:
'Update dependencies' }  ];  limitedItems = [    { id: '1', title: 'Option A', subtitle: 'First option' },    {
id: '2', title: 'Option B', subtitle: 'Second option' },    { id: '3', title: 'Option C', subtitle: 'Third option'
},    { id: '4', title: 'Option D', subtitle: 'Fourth option' },    { id: '5', title: 'Option E', subtitle: 'Fifth
option' }  ];  onSelectionChanged(event: any) {    this.basicSelection = event;    console.log('Basic selection
changed:', event);  }  onCheckboxSelectionChanged(event: any) {    this.checkboxSelection = event;
console.log('Checkbox selection changed:', event);  }  onLimitedSelectionChanged(event: any) {
this.limitedSelection = event;    console.log('Limited selection changed:', event);  }  selectAll() {    // This
would be called on the list component reference    console.log('Select all clicked');  }  clearSelection() {    //
This would be called on the list component reference    console.log('Clear selection clicked');  }
selectSpecific() {    // This would be called on the list component reference    console.log('Select specific
items clicked');  }}
```

Multi-Selection Features

• Checkbox Mode: Visual checkboxes for clear selection indication

• Selection Limits: Set maximum number of selectable items

• Programmatic Control:selectAll(),clearSelection(), andselectItems()methods

• Event Handling: Comprehensive selection change events with detailed information

```
selectAll()
```

```
clearSelection()
```

```
selectItems()
```

Accessibility

WCAG 2.1 AA compliant with comprehensive keyboard navigation and screen reader support.

Accessibility Features

• Keyboard Navigation: Full keyboard support with arrow keys, tab, enter, and escape

• ARIA Support: Comprehensive ARIA labels, roles, and state announcements

• Screen Reader: Descriptive labels and status announcements

• Focus Management: Clear visual focus indicators and logical tab order

• High Contrast: Enhanced visibility in high contrast modes

• Testing Checklist: Complete accessibility testing guidelines

Component Architecture

The List component consists of two main parts:

AavaListComponent

The main list container that handles selection, validation, and form integration.

AavaListItemsComponent

Individual list item wrapper with content projection slots and styling.

```
import { AavaListItemsComponent } from "@aava/play-core";@Component({  selector: "app-list-example",  template: `    <aava-list [items]="items" [multiSelect]="true">      <aava-list-items        *ngFor="let item of items"        [selected]="isSelected(item)"        [disabled]="item.disabled"        [size]="'md'"        (itemClick)="onItemClick(item)"      >        <!-- Left slot for avatar/icon -->        <div left>          <aava-avatars [imageUrl]="item.avatar?.imageUrl"></aava-avatars>        </div>        <!-- Middle slot for content -->        <div middle>          <h3>{{ item.title }}</h3>          <p>{{ item.subtitle }}</p>        </div>        <!-- Right slot for actions -->        <div right>          <aava-button            [label]="'Edit'"            (userClick)="onEdit(item)"          ></aava-button>        </div>      </aava-list-items>    </aava-list>  `,})export class ListExampleComponent {  // Component implementation}
```

```
import { AavaListItemsComponent } from "@aava/play-core";@Component({  selector: "app-list-example",  template: `    <aava-list [items]="items" [multiSelect]="true">      <aava-list-items        *ngFor="let item of items"        [selected]="isSelected(item)"        [disabled]="item.disabled"        [size]="'md'"        (itemClick)="onItemClick(item)"      >        <!-- Left slot for avatar/icon -->        <div left>          <aava-avatars [imageUrl]="item.avatar?.imageUrl"></aava-avatars>        </div>        <!-- Middle slot for content -->        <div middle>          <h3>{{ item.title }}</h3>          <p>{{ item.subtitle }}</p>        </div>        <!-- Right slot for actions -->        <div right>          <aava-button            [label]="'Edit'"            (userClick)="onEdit(item)"          ></aava-button>        </div>      </aava-list-items>    </aava-list>  `,})export class ListExampleComponent {  // Component implementation}
```

Content Projection Slots

• [left]: For avatars, icons, or left-aligned content

• [middle]: For main content like titles and subtitles

• [right]: For action buttons or right-aligned content

• Default slot: For any additional content

```
[left]
```

```
[middle]
```

```
[right]
```

API Reference

Inputs

```
Property | Type | Default | Description
title | string | '' | Title displayed above the list
items | ListItem[] | [] | Array of list items to display
height | string | '400px' | Height of the list container
width | string | '100%' | Width of the list container
emptyLabel | string | 'No items available' | Text displayed when list is empty
multiSelect | boolean | false | Enable multi-selection mode
maxSelections | number | undefined | Maximum number of items that can be selected
selectedItemId | string | null | null | Currently selected item ID (single select)
selectedItemIds | string[] | [] | Array of selected item IDs (multi-select)
showCheckboxes | boolean | false | Show checkboxes for multi-selection
selectionMode | 'click' | 'checkbox' | 'click' | Selection interaction mode
required | boolean | false | Whether the list selection is required
errorMessage | string | 'Please select at least one item' | Custom error message
errorPosition | 'top' | 'bottom' | 'bottom' | Position of error message
showErrorImmediately | boolean | true | Show error immediately or wait for touch
```

```
title
```

```
string
```

```
''
```

```
items
```

```
ListItem[]
```

```
[]
```

```
height
```

```
string
```

```
'400px'
```

```
width
```

```
string
```

```
'100%'
```

```
emptyLabel
```

```
string
```

```
'No items available'
```

```
multiSelect
```

```
boolean
```

```
false
```

```
maxSelections
```

```
number
```

```
undefined
```

```
selectedItemId
```

```
string | null
```

```
null
```

```
selectedItemIds
```

```
string[]
```

```
[]
```

```
showCheckboxes
```

```
boolean
```

```
false
```

```
selectionMode
```

```
'click' | 'checkbox'
```

```
'click'
```

```
required
```

```
boolean
```

```
false
```

```
errorMessage
```

```
string
```

```
'Please select at least one item'
```

```
errorPosition
```

```
'top' | 'bottom'
```

```
'bottom'
```

```
showErrorImmediately
```

```
boolean
```

```
true
```

ListItemsComponent Inputs

```
Property | Type | Default | Description
selected | boolean | false | Whether the item is selected
disabled | boolean | false | Whether the item is disabled
outline | boolean | false | Whether to show outline styling
size | ListItemSize | 'md' | Size variant (xs, sm, md, lg)
```

```
selected
```

```
boolean
```

```
false
```

```
disabled
```

```
boolean
```

```
false
```

```
outline
```

```
boolean
```

```
false
```

```
size
```

```
ListItemSize
```

```
'md'
```

Outputs

```
Event | Type | Description
onOptionSelected | EventEmitter | Emitted when an item is selected
onSelectionChanged | EventEmitter | Emitted when selection changes
onButtonClick | EventEmitter | Emitted when an item button is clicked
onIconClick | EventEmitter<{item: ListItem, event: Event}> | Emitted when an item icon is clicked
```

```
onOptionSelected
```

```
EventEmitter<ListItem>
```

```
onSelectionChanged
```

```
EventEmitter<ListSelectionEvent>
```

```
onButtonClick
```

```
EventEmitter<ListButtonClickEvent>
```

```
onIconClick
```

```
EventEmitter<{item: ListItem, event: Event}>
```

ListItemsComponent Outputs

```
Event | Type | Description
itemClick | EventEmitter | Emitted when the item is clicked
```

```
itemClick
```

```
EventEmitter<void>
```

Properties

```
Property | Type | Description
value | string | string[] | null | Current form value (getter/setter)
disabled | boolean | Whether the component is disabled
touched | boolean | Whether the component has been touched
hasError | boolean | Whether the component has validation errors
```

```
value
```

```
string | string[] | null
```

```
disabled
```

```
boolean
```

```
touched
```

```
boolean
```

```
hasError
```

```
boolean
```

Methods

```
Method | Parameters | Return | Description
selectAll() | None | void | Select all available items (multi-select only)
clearSelection() | None | void | Clear all selections
selectItems(itemIds: string[]) | itemIds: string[] | void | Select specific items by ID
validate() | None | boolean | Manually trigger validation
resetValidation() | None | void | Reset validation state
hideErrorImmediately() | None | void | Hide error message immediately
showErrorImmediatelyMethod() | None | void | Show error message immediately
trackByFn(index: number, item: ListItem) | index: number, item: ListItem | string | Track function for efficient
rendering
trackByButtonFn(index: number, button: ListItemButton) | index: number, button: ListItemButton | string | Track
function for button rendering
onItemClick(item: ListItem, event?: Event) | item: ListItem, event?: Event | void | Handle item click events
onCheckboxChange(item: ListItem, event: Event) | item: ListItem, event: Event | void | Handle checkbox change
events
onItemButtonClick(item: ListItem, button: ListItemButton, buttonIndex: number, event: Event) | item: ListItem,
button: ListItemButton, buttonIndex: number, event: Event | void | Handle button click events
onItemIconClick(item: ListItem, event: Event) | item: ListItem, event: Event | void | Handle icon click events
hasAvatar(item: ListItem) | item: ListItem | boolean | Check if item has avatar
hasIcon(item: ListItem) | item: ListItem | boolean | Check if item has icon
hasButtons(item: ListItem) | item: ListItem | boolean | Check if item has buttons
isIconClickable(item: ListItem) | item: ListItem | boolean | Check if icon is clickable
canSelectMore() | None | boolean | Check if more items can be selected
isEmpty() | None | boolean | Check if selection is empty
isClickOnActionElement(event: Event) | event: Event | boolean | Check if click is on action element
```

```
selectAll()
```

```
void
```

```
clearSelection()
```

```
void
```

```
selectItems(itemIds: string[])
```

```
itemIds: string[]
```

```
void
```

```
validate()
```

```
boolean
```

```
resetValidation()
```

```
void
```

```
hideErrorImmediately()
```

```
void
```

```
showErrorImmediatelyMethod()
```

```
void
```

```
trackByFn(index: number, item: ListItem)
```

```
index: number, item: ListItem
```

```
string
```

```
trackByButtonFn(index: number, button: ListItemButton)
```

```
index: number, button: ListItemButton
```

```
string
```

```
onItemClick(item: ListItem, event?: Event)
```

```
item: ListItem, event?: Event
```

```
void
```

```
onCheckboxChange(item: ListItem, event: Event)
```

```
item: ListItem, event: Event
```

```
void
```

```
onItemButtonClick(item: ListItem, button: ListItemButton, buttonIndex: number, event: Event)
```

```
item: ListItem, button: ListItemButton, buttonIndex: number, event: Event
```

```
void
```

```
onItemIconClick(item: ListItem, event: Event)
```

```
item: ListItem, event: Event
```

```
void
```

```
hasAvatar(item: ListItem)
```

```
item: ListItem
```

```
boolean
```

```
hasIcon(item: ListItem)
```

```
item: ListItem
```

```
boolean
```

```
hasButtons(item: ListItem)
```

```
item: ListItem
```

```
boolean
```

```
isIconClickable(item: ListItem)
```

```
item: ListItem
```

```
boolean
```

```
canSelectMore()
```

```
boolean
```

```
isEmpty()
```

```
boolean
```

```
isClickOnActionElement(event: Event)
```

```
event: Event
```

```
boolean
```

Interfaces

ListItem

```
interface ListItem {  id: string;  title: string;  subtitle?: string;  avatar?: {    imageUrl?: string;    size?: AvatarSize;    shape?: AvatarShape;    statusText?: string;    profileText?: string;    badgeState?: BadgeState;    badgeSize?: BadgeSize;    badgeCount?: number;    active?: boolean;    processedanddone?: boolean;  };  icon?: {    iconName: string;    color?: string;    iconColor?: string;    iconSize?: number | string;    disabled?: boolean;    cursor?: boolean;  };  buttons?: ListItemButton[];  disabled?: boolean;  data?: any;}
```

```
interface ListItem {  id: string;  title: string;  subtitle?: string;  avatar?: {    imageUrl?: string;    size?: AvatarSize;    shape?: AvatarShape;    statusText?: string;    profileText?: string;    badgeState?: BadgeState;    badgeSize?: BadgeSize;    badgeCount?: number;    active?: boolean;    processedanddone?: boolean;  };  icon?: {    iconName: string;    color?: string;    iconColor?: string;    iconSize?: number | string;    disabled?: boolean;    cursor?: boolean;  };  buttons?: ListItemButton[];  disabled?: boolean;  data?: any;}
```

ListItemButton

```
interface ListItemButton {  label?: string;  variant?: ButtonVariant;  size?: ButtonSize;  iconName?: string;  iconColor?: string;  iconSize?: number;  iconPosition?: "left" | "right" | "only";  disabled?: boolean;  processing?: boolean;  pill?: boolean;  width?: string;  height?: string;  id?: string;  data?: any;}
```

```
interface ListItemButton {  label?: string;  variant?: ButtonVariant;  size?: ButtonSize;  iconName?: string;  iconColor?: string;  iconSize?: number;  iconPosition?: "left" | "right" | "only";  disabled?: boolean;  processing?: boolean;  pill?: boolean;  width?: string;  height?: string;  id?: string;  data?: any;}
```

ListSelectionEvent

```
interface ListSelectionEvent {  selectedItems: ListItem[];  selectedIds: string[];  lastSelectedItem: ListItem;}
```

```
interface ListSelectionEvent {  selectedItems: ListItem[];  selectedIds: string[];  lastSelectedItem: ListItem;}
```

ListButtonClickEvent

```
interface ListButtonClickEvent {  item: ListItem;  button: ListItemButton;  buttonIndex: number;  event: Event;}
```

```
interface ListButtonClickEvent {  item: ListItem;  button: ListItemButton;  buttonIndex: number;  event: Event;}
```

ListItemSize

```
type ListItemSize = "xs" | "sm" | "md" | "lg";
```

```
type ListItemSize = "xs" | "sm" | "md" | "lg";
```

Design Tokens & Theming

AAVA Play List uses semantic design tokens for all surfaces, spacing, radius, and motion. The component exposes scoped override tokens for fine-tuning appearance while maintaining design system consistency.

Available Design Tokens for List

```
Token | Purpose | Default Value
--list-container-border-radius | Border radius of list container | Theme-based
--list-container-padding | Padding inside list container | Theme-based
--list-container-gap | Gap between list elements | Theme-based
--list-container-border | Border style for list container | Theme-based
--list-background-color | Background color of list | Theme-based
```

```
--list-container-border-radius
```

```
--list-container-padding
```

```
--list-container-gap
```

```
--list-container-border
```

```
--list-background-color
```

```
Token | Purpose | Default Value
--list-title-color | Color for list title | Theme-based
--list-title-size | Font size for list title | Theme-based
--list-title-weight | Font weight for list title | Theme-based
--list-title-font-family | Font family for list title | Theme-based
--list-item-color | Color for list item text | Theme-based
--list-item-subtitle-color | Color for subtitle text | Theme-based
```

```
--list-title-color
```

```
--list-title-size
```

```
--list-title-weight
```

```
--list-title-font-family
```

```
--list-item-color
```

```
--list-item-subtitle-color
```

```
Token | Purpose | Default Value
--list-items-gap | Gap between list items | Theme-based
--list-item-gap | Gap within list item elements | Theme-based
--list-item-padding | Padding inside list items | Theme-based
--list-item-border-radius | Border radius of list items | Theme-based
--list-item-background | Background color of list items | Theme-based
--list-item-border-color | Border color for list items | Theme-based
```

```
--list-items-gap
```

```
--list-item-gap
```

```
--list-item-padding
```

```
--list-item-border-radius
```

```
--list-item-background
```

```
--list-item-border-color
```

```
Token | Purpose | Default Value
--list-item-active-border | Border style for selected items | Theme-based
--list-active-bg | Background color for selected items | Theme-based
--list-buttons-gap | Gap between action buttons | Theme-based
```

```
--list-item-active-border
```

```
--list-active-bg
```

```
--list-buttons-gap
```

```
Token | Purpose | Default Value
```

```
--list-error-text | Color for error messages | Theme-based
--list-error-font-size | Font size for error text | Theme-based
--list-disable-color | Color for disabled elements | Theme-based
```

```
--list-error-text
```

```
--list-error-font-size
```

```
--list-disable-color
```

Token Override Example

```
/* Custom list theming */.my-compact-list {  --list-item-padding: 8px 12px;  --list-items-gap: 4px;  --list-container-border-radius: 4px;}.my-dense-list {  --list-title-size: 14px;  --list-item-gap: 8px;  --list-buttons-gap: 4px;}
```

```
/* Custom list theming */.my-compact-list {  --list-item-padding: 8px 12px;  --list-items-gap: 4px;  --list-container-border-radius: 4px;}.my-dense-list {  --list-title-size: 14px;  --list-item-gap: 8px;  --list-buttons-gap: 4px;}
```

Best Practices

Design Guidelines

• Content Structure: Use clear, descriptive titles and subtitles for better scanability

• Avatar Usage: Provide meaningful avatar content (images or initials) for user identification

• Icon Integration: Use appropriate icons that enhance understanding without cluttering

• Action Buttons: Limit the number of action buttons per item to maintain clean interface

• Selection Patterns: Choose single selection for mutually exclusive choices, multi-selection for independent choices

Component Architecture

• Content Projection: Use theaava-list-itemscomponent for consistent item rendering

• Slot System: Utilize left, middle, right, and default slots for flexible content layout

• Event Handling: Properly handle click events to avoid conflicts between item selection and button actions

• Performance: Use trackBy functions for efficient rendering of large lists

• State Management: Leverage the built-in selection state management for consistent behavior

```
aava-list-items
```

Accessibility

• Clear Labeling: Ensure all interactive elements have descriptive, meaningful labels

• Keyboard Navigation: Test complete keyboard navigation flow including arrow keys and activation

• Screen Reader Support: Verify proper announcement of selection changes and item states

• Color Contrast: Maintain sufficient contrast for all text and interactive elements

• Focus Management: Provide clear visual focus indicators and logical tab order

Performance

• OnPush Strategy: Component uses OnPush change detection for optimal performance

• TrackBy Functions: Efficient rendering with custom trackBy functions for large lists

• Lazy Loading: Consider lazy loading patterns for very large datasets

- Virtual Scrolling: Implement virtual scrolling for lists with hundreds or thousands of items
- Event Optimization: Debounce rapid selection changes and optimize event handlers
- Memory Management: Automatic cleanup of event listeners and references
- Rendering Optimization: Conditional rendering based on item properties and states

Form Integration

- Validation Strategy: Always validate required selections with clear error messages
- Form Patterns: Use reactive forms for complex validation scenarios
- Default Values: Set appropriate default selections for better user experience
- Reset Behavior: Define clear reset and initial state behavior for forms
- Cross-Field Validation: Implement proper validation relationships between form fields
- ControlValueAccessor: Full implementation for seamless form integration
- Touch Management: Automatic touch state management for validation timing
- Error Display Control: Programmatic control over error message visibility