

BUTTON

The Button is one of the most commonly used fundamental components of AAVA Play. It is not just a click target. It's the pulse of the interface—alive, intentional, and radiant with purpose.

The

```
<aava-button>
```

component provides a highly customizable clickable element with advanced visual effects including glass morphism, multiple interaction states, and comprehensive theming options. It supports various variants, sizes, icons, and custom styling while maintaining accessibility standards.

How to use

```
import {  
  AavaButtonComponent  
}  
from "@aava/play-core"  
;
```

Interactive Matrix

Explore all button combinations with our interactive playground.

```
<div class="matrix-demo">  
  <div class="demo-header">  
    <!-- Matrix Table -->  
    <div class="matrix-table-container">  
      <!-- Size Headers -->  
      <div id="size-tabs" class="size-tabs">  
        <aava-tabs  
          [tabs]="sizeTabs"  
          [activeTabId]="selectedSize"  
          variant="button"  
          size="sm"
```

```
[showContentPanels]="false"
(tabChange)="onSizeTabChange($event)"
class="size-tabs-container"
></aava-tabs>
</div>
<!-- Matrix Grid -->
<div class="matrix-grid">
<!-- Column Headers -->
<div class="matrix-header">
<div class="mode-header">Mode</div>
<div class="fill-header" *ngFor="let fill of fills">{{ fill }}</div>
</div>
<!-- Matrix Rows -->
<div class="matrix-row" *ngFor="let mode of modes">
<div class="mode-label">{{ mode }}</div>
<div class="button-cell" *ngFor="let fill of fills">
<ng-container
*nglf="getButtonConfig(mode, fill, selectedSize) as config"
>
<aava-button
*nglf="config.available"
[label]="config.label"
[pill]="config.pill"
[outlined]="config.outlined"
[clear]="config.clear"
[size]="getSizeMapping(selectedSize)"
[iconName]="config.iconName"
[iconPosition]="config.iconPosition || 'left'"
variant="primary"
```

```
class="matrix-button"
></aava-button>
<div *ngIf="!config.available" class="unavailable">
  ■ Not Available
</div>
</ng-container>
</div>
</div>
</div>
</div>
</div>
</div>
</div>
</div>
---
type ButtonMode = 'pill' | 'default' | 'action' | 'quick-action';
type ButtonFill = 'filled' | 'outline' | 'clear';
type ButtonSize = 'xs' | 'sm' | 'md' | 'lg' | 'xl';
interface ButtonConfig {
  mode: ButtonMode;
  fill: ButtonFill;
  size: ButtonSize;
  pill: boolean;
  outlined: boolean;
  clear: boolean;
  iconPosition?: 'left' | 'right' | 'only';
  iconName: string;
  label: string;
  available: boolean;
  iconColor?: string;
}
```

```
modes: ButtonMode[] = ['pill', 'default', 'action', 'quick-action'];

fills: ButtonFill[] = ['filled', 'outline', 'clear'];

sizes: ButtonSize[] = ['xsmall', 'small', 'medium', 'large', 'xlarge'];

sizeTabs: TabItem[] = [
  { id: 'xsmall', label: 'XSmall' },
  { id: 'small', label: 'Small' },
  { id: 'medium', label: 'Medium' },
  { id: 'large', label: 'Large' },
  { id: 'xlarge', label: 'XLarge' },
];

variants: ButtonVariant[] = [
  'primary',
  'secondary',
  'success',
  'warning',
  'danger',
  'info',
];

selectedSize: ButtonSize = 'md';

selectedMode: ButtonMode = 'pill';

selectedFill: ButtonFill = 'filled';

toggleTheme(): void {
  this.isDarkTheme = !this.isDarkTheme;
  document.body.setAttribute(
    'data-theme',
    this.isDarkTheme ? 'dark' : 'light'
  );
}

onSizeTabChange(tab: TabItem): void {
```

```
this.selectedSize = tab.id as ButtonSize;  
}  
  
getSizeLabel(size: ButtonSize): string {  
    switch (size) {  
        case 'xs':  
            return 'XSmall';  
        case 'sm':  
            return 'Small';  
        case 'md':  
            return 'Medium';  
        case 'lg':  
            return 'Large';  
        case 'xl':  
            return 'XLarge';  
        default:  
            return size;  
    }  
}  
  
getSizeMapping(  
    size: ButtonSize  
) : 'xs' | 'sm' | 'md' | 'lg' | 'xl' {  
    switch (size) {  
        case 'xs':  
            return 'xs';  
        case 'sm':  
            return 'sm';  
        case 'md':  
            return 'md';  
        case 'lg':  
            return 'lg';  
    }  
}
```

```
return 'lg';

case 'xl':
return 'xl';

default:
return 'md';
}

}

getButtonConfig(
mode: ButtonMode,
fill: ButtonFill,
size: ButtonSize
): ButtonConfig {
// All fills are now available
const baseConfig = {
mode,
fill,
size,
outlined: fill === 'outline',
clear: fill === 'clear',
available: true,
iconColor: '#fff',
};

switch (mode) {
case 'pill':
return {
...baseConfig,
pill: true,
iconPosition: 'left' as const,
iconName: 'star',

```

```
label: 'Pill',  
};  
case 'default':  
return {  
...baseConfig,  
pill: false,  
iconName: "",  
iconPosition: 'left' as const,  
label: 'Default',  
};  
case 'action':  
return {  
...baseConfig,  
pill: false,  
iconPosition: 'left' as const,  
iconName: 'zap',  
label: 'Action',  
};  
case 'quick-action':  
return {  
...baseConfig,  
pill: true,  
iconPosition: 'only' as const,  
iconName: 'plus',  
label: "",  
};  
default:  
return {  
...baseConfig,
```

```
pill: false,  
iconPosition: undefined,  
iconName: "",  
label: mode,  
available: false,  
};  
}  
}
```

Basic Usage

Basic button implementation with default settings.

```
<aava-button  
label="Primary"  
variant="primary"  
(userClick)="onButtonClick($event)"  
[pill]="true"  
>  
</aava-button>  
<aava-button  
label="Primary"  
variant="primary"  
(userClick)="onButtonClick($event)"  
>  
</aava-button>  
---  
onButtonClick(event: Event) {  
  console.log('Button clicked:', event);  
}
```

Variants

The button component supports 9 semantic variants that control the visual appearance and meaning of the button.

```
<aava-button label="Primary" variant="primary"></aava-button>  
<aava-button label="Secondary" variant="secondary"></aava-button>  
<aava-button label="Success" variant="success"></aava-button>  
<aava-button label="Warning" variant="warning"></aava-button>  
<aava-button label="Danger" variant="danger"></aava-button>  
<aava-button label="Info" variant="info"></aava-button>  
<avaa-button label="Tertiary" variant="tertiary"></avaa-button>
```

Available Variants

- primary- Main call-to-action button (pink/brand color)
- secondary- Outlined style with transparent background
- success- Positive actions (green)
- warning- Cautionary actions (orange)
- danger- Destructive actions (red)
- info- Informational actions (blue)
- tertiary- Text-only action (transparent)

Sizes

Five size options to fit different layout requirements and visual hierarchies.

```
<aava-button label="Extra Small" variant="primary" size="xs"></aava-button>  
<aava-button label="Small" variant="primary" size="sm"></aava-button>  
<aava-button label="Medium" variant="primary" size="md"></aava-button>  
<aava-button label="Large" variant="primary" size="lg"></aava-button>  
<aava-button label="Extra Large" variant="primary" size="xl"></aava-button>
```

Available Sizes

- xs (Extra Small)- Extra compact size for very dense interfaces
- sm (Small)- Compact size for dense interfaces
- md (Medium)- Standard size for most use cases (default)
- lg (Large)- Prominent size for primary actions

- xl (Extra Large)- Extra large size for hero sections and CTAs

Hover Effects

Dynamic hover effects that enhance user interaction feedback.

[Error extracting code]

Available Hover Effects

- torch- Internal semicircular sunrise effect (default)
- glow- Outer glow with elevation
- tint- Color overlay with brightness increase
- scale- Scale transformation with elevation

Pressed Effects

Visual feedback for button press interactions with various animation styles.

[Error extracting code]

Available Pressed Effects

- ripple- Multi-layered ripple animation (default)
- inset- Inset shadow effect

Icons

Comprehensive icon support with flexible positioning and customization options.

[Error extracting code]

Icon Properties

- iconName- Lucide icon name
- iconPosition- Position relative to text:left,right,icon-only
- iconColor- Custom icon color (defaults to currentColor)
- iconSize- Icon size in pixels (default: 20)

States

Interactive states for different user scenarios and feedback.

[Error extracting code]

Available States

- default- Programmatically active state
- processing- Loading/async operation state with pulse animation
- disabled- Non-interactive state

Shapes & Styles

Shape modifiers and style variants for different design requirements.

[Error extracting code]

Available Shapes

- default- Standard rectangular with border radius
- pill- Fully rounded corners (50px border radius)

Style Variants

- outlined- Transparent background with colored border
- clear- Transparent background with no border, uses variant text colors
- customStyles- Allows to apply inline CSS styles directly to the component.

This property accepts a key-value pair object where the key is the CSS property name and the value is the corresponding CSS value.

Events

The button component emits events for user interactions.

[Error extracting code]

Available Events

- userClick- Emitted on button click or keyboard activation (Enter/Space)

Accessibility

The button component follows WAI-ARIA accessibility guidelines:

- Proper keyboard navigation (Enter and Space key support)
- ARIA attributes for screen readers (aria-disabled,aria-pressed)
- Focus management with visible focus indicators
- Semantic button element usage

API Reference

Inputs

[TABLE]

Property	Type	Default	Description
label	string	"	Button text content
variant	ButtonVariant	'default'	Visual variant: 'default' , 'primary' , 'secondary' , 'success' , 'warning' , 'danger' , 'info'
size	ButtonSize	'md'	Button size: 'xs' , 'sm' , 'md' , 'lg' , 'xl'
state	ButtonState	'default'	Interaction state: 'default' , 'hover' , 'active' , 'disabled' , 'processing' , 'focus'
hoverEffect	ButtonHoverEffect	'torch'	Hover effect: 'torch' , 'glow' , 'tint' , 'scale' , 'none'
pressedEffect	ButtonPressedEffect	'ripple'	Pressed effect: 'ripple' , 'inset' , 'solid' , 'none'
processingEffect	ButtonProcessingEffect	'pulse'	Processing effect: 'pulse' , 'none'
focusEffect	ButtonFocusEffect	'border'	Focus effect: 'border' , 'none'
disabledEffect	ButtonDisabledEffect	'dim'	Disabled effect: 'dim' , 'none'
disabled	boolean	false	Whether button is disabled
processing	boolean	false	Whether button is in processing state
pill	boolean	false	Whether to use pill shape
outlined	boolean	false	Whether to use outlined style variant
clear	boolean	false	Whether to use clear style variant (transparent, no border)
width	string	"	Custom width value
height	string	"	Custom height value
gradient	string	undefined	Legacy – use customStyles instead
background	string	undefined	Legacy – use customStyles instead
color	string	undefined	Legacy – use customStyles instead
dropdown	boolean	false	Legacy – use separate dropdown component
glassVariant	ButtonGlassVariant	'glass-10'	Default recommended glass intensity
customStyles	Record<string, string>	{}	CSS custom properties override
iconName	string	"	Lucide icon name
iconColor	string	"	Custom icon color

iconSize | number | 20 | Icon size in pixels
iconPosition | 'left' | 'right' | 'only' | 'left' | Icon position relative to text
[/TABLE]

Outputs

[TABLE]

Event | Type | Description

userClick | EventEmitter<Event> | Emitted when button is clicked or activated via keyboard
[/TABLE]

Design Tokens & Theming

AAVA Play buttons use semantic design tokens for all surfaces, spacing, radius, and motion. While global tokens define the visual language of the system, buttons expose a set of scoped override tokens

that allow you to fine-tune appearance without breaking consistency.

Use these

only when necessary

—for instance, to adjust a button's vertical padding inside a dense UI or to sharpen the radius for compact layouts.

Available Design Tokens for Button

[TABLE]

Token | Purpose | Default Value

--button-size-xsm-padding | Padding for extra small size buttons | Theme-based
--button-size-sm-padding | Padding for small size buttons | Theme-based
--button-size-md-padding | Padding for medium size buttons | Theme-based
--button-size-lg-padding | Padding for large size buttons | Theme-based
--button-size-xlg-padding | Padding for extra large size buttons | Theme-based
--button-size-xsm-height | Height for extra small size buttons | Theme-based
--button-size-sm-height | Height for small size buttons | Theme-based
--button-size-md-height | Height for medium size buttons | Theme-based
--button-size-lg-height | Height for large size buttons | Theme-based

--button-size-xlg-height | Height for extra large size buttons | Theme-based

--button-border-radius | Border radius for default shape | Theme-based

[/TABLE]

[TABLE]

Token | Purpose | Default Value

--button-font-weight | Font weight for button text | Theme-based

--button-transition | Default transition animation | Theme-based

[/TABLE]

[TABLE]

Token | Purpose | Default Value

--button-icon-margin | Margin around icons | Theme-based

[/TABLE]

Token Override Example

You can define overrides in your theme configuration or component styles:

```
/* Custom button theming */
```

```
.my-compact-buttons
```

```
{
```

```
--button-size-md-padding
```

```
:
```

```
8
```

```
px
```

```
16
```

```
px
```

```
;
```

```
--button-border-radius
```

```
:
```

```
4
```

```
px
```

```
;  
--button-transition  
:  
100  
ms  
ease  
;  
}  
;
```

This would make buttons more compact, sharper, and snappier—ideal for dense interfaces or admin tools.

Best Practices

Design Guidelines

- Use semantic variants- Choose variants that match the action's intent (primary for main actions, danger for destructive actions)
- Default variant- Used default variant for standard buttons; primary for main CTAs
- Size selection- Use medium as the default size; extra small/extralarge for extreme cases only
- Effects system- Default effects work well together; customize only when needed
- Icon usage- Use icon-only for compact interfaces, left/right for labeled actions
- Consistent sizing- Match button sizes to surrounding elements and visual hierarchy

Accessibility

- Always provide meaningful labels- Even for icon-only buttons, use proper ARIA labels
- Keyboard navigation- Ensure all interactive elements are keyboard accessible
- Focus indicators- Maintain clear focus states for navigation
- Screen reader support- Use semantic HTML and proper ARIA attributes
- Color contrast- Ensure sufficient contrast for all variants and states

Performance

- Avoid excessive custom styling- Use built-in variants when possible
- Debounce rapid clicks- Prevent accidental multiple submissions
- Optimize icon loading- Use icon systems efficiently

- Consider bundle size- Import only needed effects and variants

TEXTBOX

The

```
<aava-textbox>
```

component provides a highly sophisticated text input element with glass morphism effects, advanced content projection system, multiple processing animations, and comprehensive Angular forms integration. It supports semantic variants, validation states, and extensive customization options.

How to use

```
import {  
    AavaTextboxComponent  
} from "@aava/play-core";
```

Basic Usage

Simple textbox implementation with label, placeholder, and two-way data binding.

```
<aava-textbox  
    label="Basic Input"  
    placeholder="Enter text here"  
    (change)="onTextboxChange($event)"></aava-textbox>  
  
---  
onTextboxChange(event: Event) {  
    const target = event.target as HTMLInputElement;  
    console.log('Textbox value changed:', target.value);  
}
```

Variants

The textbox component supports 6 semantic variants that control visual appearance and focus colors.

```
<aava-textbox  
label="Default Variant"  
placeholder="Default variant..."  
variant="default"  
(change)="onTextboxChange($event)"  
></aava-textbox>  
  
<aava-textbox  
label="Primary Variant"  
placeholder="Primary variant..."  
variant="primary"  
(change)="onTextboxChange($event)"  
></aava-textbox>  
  
<aava-textbox  
label="Success Variant"  
placeholder="Success variant..."  
variant="success"  
(change)="onTextboxChange($event)"  
></aava-textbox>  
  
<aava-textbox  
label="Error Variant"  
placeholder="Error variant..."  
variant="error"  
(change)="onTextboxChange($event)"  
></aava-textbox>  
  
<aava-textbox  
label="Warning Variant"  
placeholder="Warning variant..."  
variant="warning"  
(change)="onTextboxChange($event)"
```

```

></aava-textbox>

<aava-textbox
  label="Info Variant"
  placeholder="Info variant..."
  variant="info"
  (change)="onTextboxChange($event)"
></aava-textbox>

---

onTextboxChange(event: Event) {
  const target = event.target as HTMLInputElement;
  console.log('Textbox value changed:', target.value);
}

```

Available Variants

- default- Standard neutral appearance with brand primary focus
- primary- Primary variant with enhanced brand focus color
- success- Positive states and confirmations (green)
- error- Error states and validation failures (red)
- warning- Warning states and cautions (orange/yellow)
- info- Informational states and tips (blue)

Sizes

Five size options to accommodate different interface densities and layout requirements.

```

<aava-textbox
  label="Extra Small"
  placeholder="Extra Small..."
  size="xs"
></aava-textbox>

<aava-textbox label="Small" placeholder="Small..." size="sm"></aava-textbox>
<aava-textbox label="Medium" placeholder="Medium..." size="md"></aava-textbox>

```

```
<aava-textbox label="Large" placeholder="Large..." size="lg"></aava-textbox>

<aava-textbox
  label="Extra Large"
  placeholder="Extra Large..."
  size="xl"
></aava-textbox>
```

Available Sizes

- xs (Extra Small)- Extra small size for very compact interfaces
- sm (Small)- Small size for dense interfaces
- md (Medium)- Medium size for most use cases (default)
- lg (Large)- Large size for prominent inputs
- xl (Extra Large)- Extra large size for emphasis and accessibility

Icons & Affixes

Advanced content projection system supporting icons, prefixes, and suffixes through Angular's content projection.

```
<aava-textbox label="Amount" placeholder="0.00" type="number" (on)>
  <span slot="prefix">$</span>
  <span slot="suffix">USD</span>
</aava-textbox>

<aava-textbox
  label="Password"
  [type]="showPassword ? 'text' : 'password'"
  placeholder="Enter password"
>
  <aava-icon
    slot="icon-end"
    [iconName]="showPassword ? 'eye' : 'eye-off'"
    (click)="togglePasswordVisibility()"
  ></aava-icon>
```

```

</aava-textbox>

<aava-textbox
    label="With Icon Separator (Start)"
    placeholder="Search..."
    [iconSeparator]="true"
    >

    <aava-icon slot="icon-start" iconName="search"></aava-icon>
    </aava-textbox>

    <aava-textbox
        label="With Icon Separator (End)"
        placeholder="Clear..."
        [iconSeparator]="true"
        >

        <aava-icon slot="icon-end" iconName="x"></aava-icon>
        </aava-textbox>

    ---
    showPassword = false;

    togglePasswordVisibility(): void {
        this.showPassword = !this.showPassword;
    }

```

Content Projection Slots

- icon-start- Icons at the beginning of the input
- icon-end- Icons at the end of the input
- prefix- Text or elements before the input text
- suffix- Text or elements after the input text

Input Masking

Advanced input masking system powered by ngx-mask for formatted input patterns like phone numbers, dates, currency, and credit cards.

<!-- Phone Number Masking -->

```
<aava-textbox
  label="Phone Number"
  placeholder="(123) 456-7890"
  [mask]="maskPhone"
>
</aava-textbox>
<!-- Currency Masking -->
<aava-textbox
  label="Currency"
  placeholder="0.00"
  [mask]="maskCurrency"
  [maskThousandSeparator]="thousand"
  [maskDecimalMarker]="decimal"
>
<span slot="prefix">$</span>
</aava-textbox>
<!-- Date Masking -->
<aava-textbox label="Date" placeholder="MM/DD/YYYY" [mask]="maskDate">
</aava-textbox>
<!-- Custom Pattern Masking -->
<aava-textbox
  label="Custom Pattern (AA-0000)"
  placeholder="AB-1234"
  [mask]="customMask"
  [maskPatterns]="customPatterns"
>
</aava-textbox>
<aava-textbox
  label="Phone Number (with Built-in Dropdown)"
```

```
[placeholder]="currentCountryPlaceholder"
[(ngModel)]="countryPhoneValue"
[mask]="currentCountryMask" [
prefixDropdown]="true"
[prefixDropdownOptions]="countryOptions"
[selectedPrefixOption]="selectedCountry"
(prefixDropdownSelect)="onCountrySelect($event)"
>
</aava-textbox>
<aava-textbox
label="Phone Number (with Custom Slot Dropdown)"
[placeholder]="slotCurrentPlaceholder"
[(ngModel)]="slotCountryPhoneValue"
[mask]="slotCurrentMask"
(clickOutSide)="closeSlotDropdown()"
>
<!-- Custom dropdown projected into prefix slot -->
<div slot="prefix" class="custom-country-dropdown">
<div class="dropdown-trigger" (click)="toggleSlotDropdown()" tabindex="0" role="button"
[attr.aria-expanded]="slotIsDropdownOpen">
<span class="country-label">{{ slotSelectedCountry.label }}</span>
<svg class="chevron-icon" [class.open]="slotIsDropdownOpen" width="16" height="16" viewBox="0 0
24 24"
fill="none" stroke="currentColor">
<polyline points="6 9 12 15 18 9"></polyline>
</svg>
</div>
</div>
<div slot="dropdown" class="custom-country-dropdown">
```

```
<div class="dropdown-menu" *ngIf="slotsDropdownOpen" role="listbox" tabindex="-1">
<div *ngFor="let option of countryOptions" class="dropdown-item"
[class.selected]="option.value === slotSelectedCountryCode"
(click)="selectSlotCountry(option); $event.stopPropagation()" (keydown.enter)="
selectSlotCountry(option); $event.stopPropagation()
" (keydown.space)="
selectSlotCountry(option); $event.stopPropagation()
" tabindex="0" role="option" [attr.aria-selected]="option.value === slotSelectedCountryCode">
{{ option.label }}
</div>
</div>
</div>
</aava-textbox>
---
maskPhone = '(000) 000-0000';
maskCurrency = 'separator.2';
thousand: ',' | '' = ',';
decimal: '.' | ',' | ['.','.'] = '.';
maskDate = '00/00/0000';
customMask = 'SS-0000';
customPatterns: Record<string, { pattern: RegExp }> = {
  S: { pattern: /[A-Za-z]/ },
  '0': { pattern: /\d/ },
};
countryPhoneValue = "";
countryOptions = [
{
  label: 'US +1',
  value: 'us',
```

```
        mask: '(000) 000-0000',
        placeholder: '(123) 456-7890',
      },
    {
      label: 'IN +91',
      value: 'in',
      mask: '00000 00000',
      placeholder: '98765 43210',
    },
    {
      label: 'UK +44',
      value: 'uk',
      mask: '00000 000000',
      placeholder: '20123 456789',
    },
  ];
  selectedCountry = this.countryOptions[0];
  get currentCountryMask(): string {
    return this.selectedCountry.mask;
  }
  get currentCountryPlaceholder(): string {
    return this.selectedCountry.placeholder;
  }
  onCountrySelect(option: { label: string; value: string }): void {
    const country = this.countryOptions.find((c) => c.value === option.value);
    if (country) {
      this.selectedCountry = country;
      this.countryPhoneValue = "";
    }
  }
}
```

```
}

slotCountryPhoneValue = "";

slotSelectedCountryCode = 'us';

slotIsDropdownOpen = false;

get slotSelectedCountry() {

    return (

        this.countryOptions.find(
            (c) => c.value === this.slotSelectedCountryCode
        ) || this.countryOptions[0]
    );
}

get slotCurrentMask(): string {

    return this.slotSelectedCountry.mask;
}

get slotCurrentPlaceholder(): string {

    return this.slotSelectedCountry.placeholder;
}

toggleSlotDropdown(): void {

    this.slotIsDropdownOpen = !this.slotIsDropdownOpen;
}

selectSlotCountry(option: {

    label: string;
    value: string;
    mask: string;
    placeholder: string;
}): void {

    this.slotSelectedCountryCode = option.value;
    this.slotIsDropdownOpen = false;
    this.slotCountryPhoneValue = "";
}
```

```
}

closeSlotDropdown(): void {
  if (this.slotIsDropdownOpen) {
    this.slotIsDropdownOpen = false;
  }
}
```

Masking Features

- Pattern-Based Input: Define custom input patterns using mask syntax
- Real-time Formatting: Automatic formatting as users type
- Special Character Handling: Control how special characters are processed
- Prefix/Suffix Support: Add currency symbols, units, or other prefixes/suffixes
- Validation Integration: Works seamlessly with existing validation system
- Accessibility: Maintains proper ARIA attributes and keyboard navigation

Common Mask Patterns

- Phone Numbers:(000) 000-0000for US format
- Dates:00/00/0000for MM/DD/YYYY format
- Currency:separator.2with thousand separators and decimal places
- Credit Cards:0000 0000 0000 0000for card number format
- Custom Patterns: Define your own patterns using0,9,A,placeholders

States & Validation

Comprehensive validation system with error messages, helper text, and various input states.

[Error extracting code]

Available States

- Normal- Default input state
- Focused- Active input with enhanced border and shadow
- Disabled- Non-interactive state with dimmed appearance
- Readonly- Display-only state with modified styling
- Error- Validation error state with red styling and error message

- Required- Indicates mandatory fields with asterisk

Validation Features

Error Handling:

- Error messages- Display validation errors with icon
- ARIA compliance- Proper aria-invalid and aria-describedby attributes
- Visual feedback- Error variant styling with red colors
- Icon integration- Alert icons automatically shown with errors

Helper Text:

- Guidance messages- Helpful instructions below input
- Icon support- Info icons automatically shown with helper text
- Conditional display- Helper text hidden when errors are present
- Accessibility- Proper ARIA relationships

Required Fields:

- Visual indicators- Asterisk (*) displayed for required fields
- Label integration- Required indicator integrated with label
- Form validation- Works with Angular form validation

Processing Effects

Advanced processing states with multiple animation options for loading and async operations.

[Error extracting code]

Available Processing Effects

- border-pulse- Pulsing border animation (default)
- shimmer- Text shimmer effect within input
- gradient-border- Animated multi-color border gradient

Effects System

Modern effects system following Text Input Specifications for consistent visual behavior and accessibility.

Hover Effects

- tint- Subtle color tinting on hover (recommended)

- glow- Enhanced glow effect on hover (for interactive elements)

Processing States

- Default Processing- Border pulse animation with customizable colors
- Shimmer Effect- Text shimmer animation as alternative to border pulse
- Gradient Border- Animated multi-color gradient border for processing state
- Custom Colors- Configurable gradient colors for brand consistency

Accessibility Features

- High Contrast Support- Effects respect high contrast mode settings
- Reduced Motion- Animations respect user's motion preferences
- Focus Indicators- Clear focus states maintained with all effects
- Screen Reader Support- Proper ARIA attributes for all interactive states

API Reference

Inputs

[TABLE]

Property | Type | Default | Description

label | string | " | Visible label text above input

placeholder | string | " | Placeholder text shown when empty

variant | TextboxVariant | 'default' | Visual variant: | 'default' | , | 'primary' | , | 'success' | , | 'error' | , | 'warning' | , | 'info'

size | TextboxSize | 'md' | Input size: | 'xs' | , | 'sm' | , | 'md' | , | 'lg' | , | 'xl'

disabled | boolean | false | Whether input is disabled

readonly | boolean | false | Whether input is read-only

error | string | " | Error message to display

helper | string | " | Helper text to display

required | boolean | false | Whether input is required

fullWidth | boolean | false | Whether input takes full container width

type | string | 'text' | HTML input type

maxlength | number | " | Maximum character length

minlength | number | " | Minimum character length
autocomplete | string | " | HTML autocomplete attribute
id | string | " | Custom element ID
name | string | " | HTML name attribute
icon | string | " | Icon name to display
iconPosition | 'start' | 'end' | 'start' | Position of the icon relative to input
iconSeparator | boolean | false | Whether to show separator between icon and input
iconSpacing | 'compact' | 'normal' | 'relaxed' | 'normal' | Icon spacing variant
inputKind | 'text' | 'phone' | 'currency' | 'password' | 'text' | Type of input for specialized behavior
inputKindLabel | string | " | Label for specialized input types (e.g., country code)
phone | boolean | false | Enable phone input functionality
labelPosition | 'start' | 'end' | 'start' | Position of country prefix label for phone inputs
[/TABLE]

Masking Properties

[TABLE]

Property | Type | Default | Description

mask | string | null | null | Input mask pattern (e.g., "(000) 000-0000" for phone)
maskPrefix | string | " | Prefix to add before masked value (e.g., "\$" for currency)
maskSuffix | string | " | Suffix to add after masked value (e.g., "%" for percentage)
maskDropSpecialCharacters | boolean | string[] | true | Whether to drop special characters from value
maskShowMaskTyped | boolean | false | Show mask characters as user types
maskThousandSeparator | string | " | Thousand separator character (e.g., "," for numbers)
maskDecimalMarker | '.' | ',' | ['.',''] | ' ' | Decimal marker character
maskPatterns | Record<string, object> | {} | Custom mask patterns for special characters
maskValidation | boolean | false | Enable mask validation
maskAllowNegativeNumbers | boolean | false | Allow negative numbers in numeric masks
maskLeadZeroDateTime | boolean | false | Show leading zeros in date/time masks

[/TABLE]

Effects System Properties

[TABLE]

Property	Type	Default	Description
hoverEffect	'tint' 'glow' ''		Hover effect: Tint (recommended) or Glow
pressedEffect	'solid' 'solid'		Pressed effect: Solid (recommended and only allowed)
processing	boolean	false	Processing state - triggers border pulse by default
processingEffect	'shimmer'		Alternative processing effect: Text shimmer animation
processingGradientBorder	boolean	false	Show animated gradient border for processing state
processingGradientColors	string[]	['#e91e63', '#fee140', '#ff9800', '#047857', '#ff9800', '#fee140', '#e91e63']	Colors for processing gradient border
decorativeEffect	'glowBox' 'borderFlow' 'attention' 'wave'		Ambient decorative effects for future use
disabledState	'grey' 'grey'		Disabled state appearance: Grey (recommended and only allowed)
customStyles	Record<string, string>		CSS custom properties override for advanced theming

[/TABLE]

Outputs

[TABLE]

Event	Type	Description
iconStartClick	EventEmitter<Event>	Emitted when the start icon is clicked
iconEndClick	EventEmitter<Event>	Emitted when the end icon is clicked
clickOutSide	EventEmitter<boolean>	Emitted when a click occurs outside the input
change	EventEmitter<Event>	Emitted when the input value changes
blur	EventEmitter<Event>	Emitted when the input loses focus
focus	EventEmitter<Event>	Emitted when the input gains focus
input	EventEmitter<Event>	Emitted on every input event
prefixSelect	EventEmitter<{ label: string; value: string; }>	Emitted when a prefix option is selected
suffixSelect	EventEmitter<{ label: string; value: string; }>	Emitted when a suffix option is selected

[/TABLE]

Properties

[TABLE]

Property	Type	Description
value	string	Current input value
isFocused	boolean	Whether input currently has focus
hasError	boolean	Whether input has error state
hasHelper	boolean	Whether input has helper text

[/TABLE]

Methods

[TABLE]

Method	Parameters	Return Type	Description
setValue()	value: string	void	Set input value programmatically
writeValue()	value: string	void	Set input value (ControlValueAccessor)
registerOnChange()	fn: (value: string) => void	void	Register change callback
registerOnTouched()	fn: () => void	void	Register touched callback
setDisabledState()	isEnabled: boolean	void	Set disabled state

[/TABLE]

Content Projection Slots

[TABLE]

Slot	Description
icon-start	Icons displayed at the start of the input
icon-end	Icons displayed at the end of the input
prefix	Content displayed before the input text
suffix	Content displayed after the input text

[/TABLE]

CSS Custom Properties

[TABLE]

Property	Description
--textbox-glass-default-background	Background color with glass effect
--textbox-glass-default-blur	Backdrop blur amount for glass effect

--textbox-glass-default-border | Border color for default state
--textbox-glass-default-shadow | Box shadow for glass effect
--textbox-border-radius | Border radius of the input container
--textbox-transition | Transition animation duration
--textbox-input-font | Font properties for input text
--textbox-input-color | Text color for input
--textbox-input-padding | Padding inside the input
--textbox-input-min-height | Minimum height of the input
--textbox-label-font | Font properties for label
--textbox-label-color | Text color for label
--textbox-label-weight | Font weight for label
--textbox-placeholder-color | Color for placeholder text
--textbox-error-color | Color for error messages and state
--textbox-helper-color | Color for helper text
--textbox-icon-color | Color for icons in normal state
--textbox-icon-focus-color | Color for icons when focused
--textbox-variant-default | Colors for default variant
--textbox-variant-primary | Colors for primary variant
--textbox-variant-success | Colors for success variant
--textbox-variant-error | Colors for error variant
--textbox-variant-warning | Colors for warning variant
--textbox-variant-info | Colors for info variant
--textbox-size-xs-padding | Padding for extra small size
--textbox-size-sm-padding | Padding for small size
--textbox-size-md-padding | Padding for medium size
--textbox-size-lg-padding | Padding for large size
--textbox-size-xl-padding | Padding for extra large size
--textbox-size-xs-height | Height for extra small size
--textbox-size-sm-height | Height for small size

--textbox-size-md-height | Height for medium size
--textbox-size-lg-height | Height for large size
--textbox-size-xl-height | Height for extra large size

[/TABLE]

Best Practices

Design Guidelines

- Choose appropriate variants- Use semantic variants that match the context
- Provide clear labels- Always include descriptive labels for accessibility
- Use helper text wisely- Provide guidance without cluttering the interface
- Handle errors gracefully- Show clear, actionable error messages
- Optimize for touch- Ensure adequate touch targets for mobile users
- Consider glass intensity- Use glass-10 for most cases, glass-50 for emphasis
- Select appropriate sizes- Use xs for very compact layouts, xl for emphasis and accessibility
- Implement masking thoughtfully- Choose mask patterns that match user expectations and data format
- Balance effects and performance- Use effects system for enhanced UX while maintaining performance

Accessibility

- Proper labeling- Label elements correctly associated with inputs
- ARIA attributes- Use aria-invalid, aria-describedby, aria-required
- Error announcement- Error messages announced to screen readers
- Keyboard navigation- Full keyboard support for all interactions
- Focus management- Visible focus indicators and proper focus order
- Icon accessibility- Icons properly labeled and keyboard accessible

Performance

- Validate appropriately- Use client and server validation together
- Debounce input events- For real-time validation and API calls
- Optimize re-renders- Use OnPush change detection strategy
- Efficient icon handling- Load icons efficiently and cache appropriately

Form Integration

- Test with forms- Ensure proper integration with your form handling
- Handle validation- Implement comprehensive validation strategies
- Consider reset behavior- Define clear reset and initial state behavior
- Support reactive forms- Proper ControlValueAccessor implementation

Masking Best Practices

- Choose intuitive patterns- Use mask patterns that users expect (e.g., phone formats)
- Handle edge cases- Consider what happens when users paste or clear masked input
- Provide clear examples- Use placeholders that show the expected format
- Test accessibility- Ensure screen readers can properly announce masked input
- Consider internationalization- Use appropriate separators and formats for different locales
- Balance flexibility- Allow users to edit parts of masked input without losing context

TEXTAREA

A powerful multi-line text input component designed for collecting longer text content with comprehensive form integration, validation support, and accessibility features. Supports Angular Forms (reactive and template-driven), custom styling, and advanced features like processing gradient borders.

How to use

```
import {  
  AavaTextareaComponent  
}  
from "@aava/play-core"  
;
```

Basic Usage

Simple textarea implementations with labels and basic functionality.

```
<aava-textarea  
placeholder="Enter your text here..."  
[rows]="4"  
(change)="onTextareaChange($event)"  
[processing]="true"  
variant="primary"  
></aava-textarea>  
  
---  
onTextareaChange(event: Event) {  
  const target = event.target as HTMLTextAreaElement;  
  console.log('Textarea value changed:', target.value);  
}
```

Sizes

Three size variants to accommodate different interface requirements and content needs.

```
<aava-textarea
  label="Extra Small"
  placeholder="Extra small textarea..."
  size="xs"
  [rows]="2"
  (change)="onTextareaChange($event)"
></aava-textarea>

<aava-textarea
  label="Small"
  placeholder="Small textarea..."
  size="sm"
  [rows]="2"
  (change)="onTextareaChange($event)"
></aava-textarea>

<aava-textarea
  label="Medium"
  placeholder="Medium textarea..."
  size="md"
  [rows]="3"
  (change)="onTextareaChange($event)"
></aava-textarea>

<aava-textarea
  label="Large"
  placeholder="Large textarea..."
  size="lg"
  [rows]="4"
  (change)="onTextareaChange($event)"
></aava-textarea>

<aava-textarea
```

```
label="Extra Large"
placeholder="Extra large textarea..."
size="xl"
[rows]="4"
(change)="onTextareaChange($event)"
></aava-textarea>
---
onTextareaChange(event: Event) {
  const target = event.target as HTMLTextAreaElement;
  console.log('Textarea value changed:', target.value);
}
```

Available Sizes

- xs (Extra Small): Very compact, ideal for tight spaces or dense forms
- sm (Small): Compact for space-constrained interfaces
- md (Medium): Standard size for most use cases (default)
- lg (Large): Prominent for primary content input areas
- xl (Extra Large): Very prominent, used for standout or high-visibility inputs

Variants

Color variants for different contexts, states, and semantic meanings.

```
<aava-textarea
placeholder="Default variant..."
variant="default"
[rows]="3"
(change)="onTextareaChange($event)"
></aava-textarea>
<aava-textarea
placeholder="Primary variant..."
variant="primary"
```

```
[rows]="3"
(change)="onTextareaChange($event)"
></aava-textarea>
<aava-textarea
placeholder="Success variant..."
variant="success"
[rows]="3"
(change)="onTextareaChange($event)"
></aava-textarea>
<aava-textarea
placeholder="Error variant..."
variant="error"
[rows]="3"
(change)="onTextareaChange($event)"
></aava-textarea>
<aava-textarea
placeholder="Warning variant..."
variant="warning"
[rows]="3"
(change)="onTextareaChange($event)"
></aava-textarea>
<aava-textarea
placeholder="Info variant..."
variant="info"
[rows]="3"
(change)="onTextareaChange($event)"
></aava-textarea>
---
onTextareaChange(event: Event) {
```

```
const target = event.target as HTMLTextAreaElement;  
console.log('Textarea value changed:', target.value);  
}
```

Style Variants

- default: Standard neutral appearance
- primary: Brand color for important inputs
- success: Green for positive confirmation states
- error: Red for validation errors and critical states
- warning: Orange for caution and attention states
- info: Blue for informational content and tips

Accessibility

Built-in accessibility features ensuring inclusive user experience.

```
<!-- Labeled Textarea -->  
  
<aava-textarea  
label="Description"  
placeholder="Enter your description..."  
[rows]="3"  
(change)="onTextareaChange($event)"  
></aava-textarea>  
  
<!-- Textarea with Helper Text -->  
  
<aava-textarea  
label="Comments"  
placeholder="Share your thoughts..."  
helper="Please provide detailed feedback to help us improve our service."  
[rows]="3"  
(change)="onTextareaChange($event)"  
></aava-textarea>  
  
<!-- Textarea with Error State -->
```

```

<aava-textarea

label="Email"

placeholder="Enter your email..."

error="Please enter a valid email address."

[rows]="3"

(change)="onTextareaChange($event)"

></aava-textarea>

<!-- Textarea with Error State -->

<aava-textarea

label="Required Field"

placeholder="This field is required..."

[required]="true"

[rows]="3"

(change)="onTextareaChange($event)"

></aava-textarea>

<!-- Textarea with Character Limit -->

<aava-textarea

label="Bio"

placeholder="Tell us about yourself (max 200 characters)..."

[maxlength]="200"

[rows]="3"

(change)="onTextareaChange($event)"

></aava-textarea>

---

onTextareaChange(event: Event) {

const target = event.target as HTMLTextAreaElement;

console.log('Textarea value changed:', target.value);

}

```

Accessibility Features

- Keyboard Navigation: Full keyboard support with proper focus management
- Screen Reader Support: Semantic HTML and ARIA attributes
- Error Announcements: Live regions for dynamic error messaging
- Focus Indicators: Clear visual focus states for navigation
- Label Association: Proper form control labeling and description
- High Contrast: Support for high contrast accessibility modes

API Reference

Inputs

[TABLE]

Property | Type | Default | Description

label | string | "" | Label text displayed above the textarea

placeholder | string | "" | Placeholder text shown when textarea is empty

variant | 'default' | 'primary' | 'success' | 'error' | 'warning' | 'info' | 'default' | Color variant for styling

size | 'xs' | , | 'sm' | , | 'md' | , | 'lg' | , | 'xl' | 'md' | Size variant of the textarea

disabled | boolean | false | Whether the textarea is disabled

readonly | boolean | false | Whether the textarea is read-only

error | string | "" | Error message to display

helper | string | "" | Helper text to provide guidance

rows | number | 3 | Number of visible text lines

id | string | "" | HTML id attribute for the textarea

name | string | "" | HTML name attribute for form submission

maxlength | number | "" | Maximum number of characters allowed

minlength | number | "" | Minimum number of characters required

required | boolean | false | Whether the field is required

fullWidth | boolean | false | Whether to take full width of container

style | Record<string, string> | "" | Custom CSS styles object

resizable | boolean | true | Whether the textarea can be resized vertically

autoResize | boolean | false | Whether to automatically resize height to fit content

processing | boolean | false | Whether to show processing state
processingGradientBorder | boolean | false | Whether to show animated gradient border
processingGradientColors | string[] | See below | Array of colors for gradient animation
[/TABLE]

Outputs

[TABLE]

Event	Type	Description
blur	EventEmitter<Event>	Emitted when the input loses focus
focus	EventEmitter<Event>	Emitted when the input gains focus
input	EventEmitter<Event>	Emitted on every input change
change	EventEmitter<Event>	Emitted when the input value changes
iconStartClick	EventEmitter<Event>	Emitted when the start icon is clicked
iconEndClick	EventEmitter<Event>	Emitted when the end icon is clicked

[/TABLE]

Properties

[TABLE]

Property	Type	Description
value	string	Current textarea value
isFocused	boolean	Whether textarea currently has focus
hasError	boolean	Whether textarea has error state
hasHelper	boolean	Whether textarea has helper text

[/TABLE]

Methods

[TABLE]

Method	Parameters	Return Type	Description
writeValue()	value: string	void	Set textarea value (ControlValueAccessor)
registerOnChange()	fn: (value: string) => void	void	Register change callback
registerOnTouched()	fn: () => void	void	Register touched callback

setDisabledState() | isEnabled: boolean | void | Set disabled state

[/TABLE]

CSS Custom Properties

[TABLE]

Property | Description

--textbox-gap | Base gap between textarea elements

--textbox-gap-xs | Gap for extra small size variant

--textbox-gap-sm | Gap for small size variant

--textbox-gap-lg | Gap for large size variant

--textbox-gap-xl | Gap for extra large size variant

--textbox-label-font | Font family for labels

--textbox-label-color | Color for label text

--textbox-label-weight | Font weight for label text

--textbox-xs-label-font-size | Label font size for extra small variant

--textbox-xs-label-weight | Label font weight for extra small variant

--textbox-xs-label-line-height | Line height for extra small label

--textbox-xs-label-font-style | Font style for extra small label

--textbox-sm-label-font-size | Label font size for small variant

--textbox-sm-label-weight | Label font weight for small variant

--textbox-sm-label-line-height | Line height for small label

--textbox-sm-label-font-style | Font style for small label

--textbox-md-label-font-size | Label font size for medium variant

--textbox-md-label-weight | Label font weight for medium variant

--textbox-md-label-line-height | Line height for medium label

--textbox-md-label-font-style | Font style for medium label

--textbox-lg-label-font-size | Label font size for large variant

--textbox-lg-label-weight | Label font weight for large variant

--textbox-lg-label-line-height | Line height for large label

--textbox-lg-label-font-style | Font style for large label

--textbox-xl-label-font-size | Label font size for extra large variant
--textbox-xl-label-weight | Label font weight for extra large variant
--textbox-xl-label-line-height | Line height for extra large label
--textbox-xl-label-font-style | Font style for extra large label
--textbox-required-color | Color for required field indicator
--textbox-textarea-container-padding | Padding around textarea container
--textbox-background | Background color of the textarea
--glass-50-border | Glass border style for textarea
--textbox-textarea-hover-primary | Primary hover color for textarea
--textbox-textarea-hover-border-width | Border width on hover
--textbox-textarea-focus-primary | Primary color for focus state
--textbox-border-error-color | Border color when there is an error
--textbox-background-disabled | Background for disabled state
--textbox-border-disabled-color | Border color for disabled state
--textbox-background-readonly | Background for readonly state
--textbox-border-readonly-color | Border color for readonly state
--textbox-xs-border-radius | Border radius for extra small variant
--textbox-sm-border-radius | Border radius for small variant
--textbox-md-border-radius | Border radius for medium variant
--textbox-lg-border-radius | Border radius for large variant
--textbox-xl-border-radius | Border radius for extra large variant
--textbox-input-xs-padding | Padding inside extra small input
--textbox-input-sm-padding | Padding inside small input
--textbox-input-md-padding | Padding inside medium input
--textbox-input-lg-padding | Padding inside large input
--textbox-input-xl-padding | Padding inside extra large input
--textbox-input-font | Font family for input text
--textbox-input-color | Text color for input
--textbox-input-padding | General input padding

--textbox-textarea-min-height | Minimum height of textarea
--textbox-textarea-min-height-xs | Minimum height for extra small variant
--textbox-textarea-min-height-sm | Minimum height for small variant
--textbox-textarea-min-height-md | Minimum height for medium variant
--textbox-textarea-min-height-lg | Minimum height for large variant
--textbox-textarea-min-height-xl | Minimum height for extra large variant
--textbox-input-xs-font-size | Font size for extra small input
--textbox-input-xs-font-weight | Font weight for extra small input
--textbox-input-xs-font-style | Font style for extra small input
--textbox-input-xs-line-height | Line height for extra small input
--textbox-input-sm-font-size | Font size for small input
--textbox-input-sm-font-weight | Font weight for small input
--textbox-input-sm-font-style | Font style for small input
--textbox-input-sm-line-height | Line height for small input
--textbox-input-md-font-size | Font size for medium input
--textbox-input-md-font-weight | Font weight for medium input
--textbox-input-md-font-style | Font style for medium input
--textbox-input-md-line-height | Line height for medium input
--textbox-input-lg-font-size | Font size for large input
--textbox-input-lg-font-weight | Font weight for large input
--textbox-input-lg-font-style | Font style for large input
--textbox-input-lg-line-height | Line height for large input
--textbox-input-xl-font-size | Font size for extra large input
--textbox-input-xl-font-weight | Font weight for extra large input
--textbox-input-xl-font-style | Font style for extra large input
--textbox-input-xl-line-height | Line height for extra large input
--textbox-affix-padding | Padding for affix elements
--textbox-affix-color | Color of affix elements
--textbox-affix-font-size | Font size of affix elements

--textbox-affix-background | Background of affix elements
--textbox-affix-border-radius | Border radius of affix elements
--textbox-affix-disabled-color | Disabled color for affix elements
--textbox-affix-disabled-background | Disabled background for affix elements
--textbox-error-gap | Gap between textarea and error text
--textbox-error-color | Color for error messages
--textbox-error-font-size | Font size for error messages
--textbox-xs-error-font-size | Font size for error text in extra small variant
--textbox-sm-error-font-size | Font size for error text in small variant
--textbox-md-error-font-size | Font size for error text in medium variant
--textbox-lg-error-font-size | Font size for error text in large variant
--textbox-xl-error-font-size | Font size for error text in extra large variant
--textbox-helper-gap | Gap between textarea and helper text
--textbox-helper-color | Color for helper text
--textbox-helper-font-size | Font size for helper text
--textbox-xs-helper-font-size | Helper font size for extra small variant
--textbox-sm-helper-font-size | Helper font size for small variant
--textbox-md-helper-font-size | Helper font size for medium variant
--textbox-lg-helper-font-size | Helper font size for large variant
--textbox-xl-helper-font-size | Helper font size for extra large variant
--textbox-border-primary-color | Primary border color
--processing-gradient-colors | Colors for gradient animation

[/TABLE]

Best Practices

Design Guidelines

- Use appropriate row heights for expected content length
- Provide clear labels and helper text for guidance
- Choose semantic variants that match content purpose

- Enable resizing for longer content input areas
- Consider processing states for form submissions

Accessibility

- Always provide meaningful labels for screen readers
- Use helper text to explain input requirements
- Ensure error messages are descriptive and actionable
- Maintain sufficient color contrast for all variants
- Test keyboard navigation and screen reader compatibility

Performance

- Use OnPush change detection strategy for better performance
- Avoid frequent style changes that trigger repaints
- Consider debouncing for real-time validation
- Use trackBy functions for dynamic textarea lists

CHECKBOX

The

```
<aava-checkbox>
```

component provides a highly interactive checkbox element with sophisticated animations, multiple visual variants, and comprehensive state management. It supports standard checked/unchecked states, indeterminate state for partial selections, and full accessibility compliance.

How to use

```
import {  
    AavaCheckboxComponent  
} from "@aava/play-core";
```

Basic Usage

Simple checkbox implementation with default settings and label.

```
<aava-checkbox  
    label="Accept terms and conditions"  
    [isChecked]="false"  
    (isCheckedChange)="onCheckboxChange($event)">  
</aava-checkbox>  
  
<aava-checkbox  
    label="Subscribe to newsletter"  
    [isChecked]="false"  
    (isCheckedChange)="onCheckboxChange($event)">  
</aava-checkbox>  
  
<aava-checkbox
```

```

label="Enable notifications"
[isChecked]="false"
(isCheckedChange)="onCheckboxChange($event)"
>
</aava-checkbox>
---
onCheckboxChange(checked: boolean) {
  console.log('Checkbox changed:', checked);
}

```

Variants

The checkbox component supports 3 distinct animation variants, each with unique visual feedback and timing.

```

<aava-checkbox
  label="Default Checked"
  variant="default"
  [isChecked]="true"
></aava-checkbox>

<aava-checkbox
  label="With-bg Checked"
  variant="with-bg"
  [isChecked]="true"
></aava-checkbox>

<aava-checkbox
  label="Animated Checked"
  variant="animated"
  [isChecked]="true"
></aava-checkbox>

```

Available Variants

- default- Clean checkbox with smooth draw/erase animations (250ms draw, 300ms erase)

- with-bg- Background fill effect with fast transitions (150ms animations)
- animated- Complex scaling background animation with staggered timing (300ms background + 300ms delay + checkmark)

Orientations

Flexible layout options for different design requirements and form arrangements.

```
<div>
```

```
  <h1>Vertical Layout (Default)</h1>
```

```
  <div>
```

```
    <aava-checkbox
```

```
      alignment="vertical"
```

```
      label="Development Tools"
```

```
      [isChecked]="false"
```

```
    ></aava-checkbox>
```

```
    <aava-checkbox
```

```
      alignment="vertical"
```

```
      label="Design Software"
```

```
      [isChecked]="true"
```

```
    ></aava-checkbox>
```

```
    <aava-checkbox
```

```
      alignment="vertical"
```

```
      label="Project Management"
```

```
      [isChecked]="false"
```

```
    ></aava-checkbox>
```

```
    <aava-checkbox
```

```
      alignment="vertical"
```

```
      label="Communication Apps"
```

```
      [isChecked]="true"
```

```
    ></aava-checkbox>
```

```
  </div>
```

```
<h1>Horizontal Layout</h1>

<div>
  <aava-checkbox
    alignment="horizontal"
    label="React"
    [isChecked]="true"
  ></aava-checkbox>
  <aava-checkbox
    alignment="horizontal"
    label="Angular"
    [isChecked]="false"
  ></aava-checkbox>
  <aava-checkbox
    alignment="horizontal"
    label="Vue.js"
    [isChecked]="true"
  ></aava-checkbox>
  <aava-checkbox
    alignment="horizontal"
    label="Svelte"
    [isChecked]="false"
  ></aava-checkbox>
  <aava-checkbox
    alignment="horizontal"
    label="Next.js"
    [isChecked]="false"
  ></aava-checkbox>
  <aava-checkbox
    alignment="horizontal"

```

```
label="Nuxt.js"
[isChecked]="false"
></aava-checkbox>
</div>
</div>
```

Available Orientations

- vertical- Default stacked layout with checkboxes arranged vertically
- horizontal- Inline layout with checkboxes arranged horizontally side by side

Sizes

Three size options to accommodate different interface densities and visual hierarchies.

```
<aava-checkbox label="Small" size="sm" variant="default" [isChecked]="false">
</aava-checkbox>

<aava-checkbox label="Medium" size="md" variant="default" [isChecked]="true">
</aava-checkbox>

<aava-checkbox label="Large" size="lg" variant="default" [isChecked]="false">
</aava-checkbox>
```

Available Sizes

- sm (Small)- 16x16px checkbox for compact interfaces
- md (Medium)- 20x20px checkbox for standard layouts (default)
- lg (Large)- 24x24px checkbox for prominent placements

States

Comprehensive state management including checked, unchecked, disabled, and indeterminate states.

```
<aava-checkbox label="Unchecked" [isChecked]="false"> </aava-checkbox>

<aava-checkbox label="Checked" [isChecked]="true"> </aava-checkbox>

<aava-checkbox label="Disabled Unchecked" [disabled]="true" [isChecked]="false">
</aava-checkbox>

<aava-checkbox label="Disabled Checked" [disabled]="true" [isChecked]="true">
</aava-checkbox>
```

```

<aava-checkbox
  label="Disabled Indeterminate"
  [disabled]="true"
  [indeterminate]="true"
>
</aava-checkbox>

<aava-checkbox label="Indeterminate" [indeterminate]="true"> </aava-checkbox>

<aava-checkbox
  label="Indeterminate (With-bg)"
  variant="with-bg"
  [indeterminate]="true"
>
</aava-checkbox>

<aava-checkbox
  label="Indeterminate (Animated)"
  variant="animated"
  [indeterminate]="true"
>
</aava-checkbox>

```

Available States

- unchecked- Default empty state
- checked- Selected state with checkmark animation
- disabled- Non-interactive state (can be checked or unchecked)
- indeterminate- Partial selection state with horizontal line indicator

State Behavior

Checked State:

- Displays animated checkmark
- Emit `true` value on change
- Visual feedback varies by variant

Disabled State:

- Non-interactive (no click/keyboard events)
- Dimmed appearance with disabled styling
- Maintains current checked/unchecked state
- Custom disabled colors via CSS tokens

Indeterminate State:

- Shows horizontal line instead of checkmark
- Used for "select all" scenarios in groups
- Clicking transitions to fully checked state
- Emit `true` when transitioning from indeterminate

Indeterminate State

Special state for representing partial selections in checkbox groups or tree structures.

[Error extracting code]

Indeterminate Usage

The indeterminate state is particularly useful for:

- Select All checkboxes- When some but not all items are selected
- Tree structures- Parent nodes with partially selected children
- Group controls- Representing mixed states in checkbox groups
- Bulk actions- Indicating partial selection in data tables

Indeterminate Behavior

- Displays horizontal line indicator instead of checkmark
- Clicking indeterminate checkbox transitions to fully checked
- Cannot be set via user interaction (must be programmatically controlled)
- Supports all variants and sizes
- Maintains proper ARIA attributes (`aria-checked="mixed"`)

Labels & Accessibility

Comprehensive accessibility features including keyboard navigation, ARIA attributes, and label support.

[Error extracting code]

Accessibility Features

Keyboard Navigation:

- Space- Toggle checkbox state
- Enter- Toggle checkbox state
- Tab/Shift+Tab- Navigate to/from checkbox

ARIA Compliance:

- role="checkbox"- Semantic role identification
- aria-checked="true|false|mixed"- Current state indication
- aria-disabled="true|false"- Disabled state indication
- aria-label- Accessibility label (falls back to visible label)

Visual Accessibility:

- High contrast focus indicators
- Proper color contrast ratios via CSS tokens
- Scalable vector icons (no pixelation)
- Motion respects user preferences

Label Behavior

- Optional labels- Checkbox works with or without visible labels
- Clickable labels- Clicking label text toggles checkbox
- Accessible labels- Always provide meaningful labels for screen readers
- Flexible positioning- Label appears to the right of checkbox

Events

Event handling for checkbox state changes and user interactions.

[Error extracting code]

Available Events

- isCheckedChange- Emitted when checkbox state changes (boolean value)

Event Details

isCheckedChange Event:

- Type: EventEmitter<boolean>
- Emitted when: User clicks checkbox or uses keyboard to toggle
- Value: true when checked, false when unchecked
- Timing: Emitted after animation completes (timing varies by variant)
- Indeterminate: Emits true when transitioning from indeterminate to checked

Event Timing by Variant

- Default: Immediate for checking, 300ms delay for unchecking
- With-bg: 150ms delay for both checking and unchecking
- Animated: 600ms delay for checking, 300ms delay for unchecking

API Reference

Inputs

[TABLE]

Property	Type	Default	Description
variant	'default' 'with-bg' 'animated'	'default'	Animation style variant
size	'sm' 'md' 'lg' 'md'	'md'	Checkbox size
alignment	'vertical' 'horizontal' 'vertical'	'vertical'	Layout orientation for checkbox groups
label	string	"	Visible label text
isChecked	boolean	false	Whether checkbox is checked
indeterminate	boolean	false	Whether checkbox is in indeterminate state
disable	boolean	false	Whether checkbox is disabled
customStyles	Record<string, string> {}		CSS custom properties override
error	string	"	Error message text
id	string	"	Unique checkbox identifier

[/TABLE]

Outputs

[TABLE]

Event	Type	Description
isCheckedChange	EventEmitter<boolean>	Emitted when checkbox state changes

isCheckChange | EventEmitter<boolean> | Emitted when checkbox state changes

[/TABLE]

Properties

[TABLE]

Property | Type | Description

showIcon | boolean | Whether to show the icon (checkmark or indeterminate)

showCheckmark | boolean | Whether to show checkmark specifically

[/TABLE]

CSS Custom Properties

[TABLE]

Property | Description

--checkbox-box-background | Background color of the checkbox box

--checkbox-box-checked-border | Border color when checked

--checkbox-box-checked-background | Background color when checked

--checkbox-box-border-disabled | Border color when disabled

--checkbox-box-background-disabled | Background color when disabled

--checkbox-icon-color-disabled | Icon color when disabled

--checkbox-box-checked-color | Checkmark color when checked

--checkbox-label-color | Text color for the label

--checkbox-label-color-disabled | Label text color when disabled

--checkbox-label-cursor | Cursor style for clickable label

--checkbox-label-cursor-disabled | Cursor style when disabled

--checkbox-cursor-disabled | Cursor style for disabled checkbox

--checkbox-size-small | Size dimensions for small variant

--checkbox-size-medium | Size dimensions for medium variant

--checkbox-size-large | Size dimensions for large variant

[/TABLE]

Animation Timing

Each variant has carefully tuned animation timing for optimal user experience:

Default Variant

- Check: 250ms cubic-bezier(0.25, 0.46, 0.45, 0.94)
- Uncheck: 300ms cubic-bezier(0.55, 0.06, 0.68, 0.19)

With-bg Variant

- Check: 150ms cubic-bezier(0.25, 0.46, 0.45, 0.94)
- Uncheck: 150ms cubic-bezier(0.55, 0.06, 0.68, 0.19)

Animated Variant

- Background: 300ms ease-out
- Checkmark: 300ms cubic-bezier(0.25, 0.46, 0.45, 0.94) with 300ms delay
- Uncheck: 300ms background + 150ms checkmark (simultaneous)

Best Practices

Design Guidelines

- Choose appropriate variants- Use default for most cases, with-bg for high contrast needs, animated for engaging interactions
- Size consistently- Match checkbox size to surrounding form elements and text size
- Group related checkboxes- Use proper form structure and fieldsets for related options
- Handle indeterminate properly- Use for partial selections in groups, not individual checkboxes

Accessibility

- Label everything- Always provide meaningful labels for accessibility
- Keyboard navigation- Ensure full keyboard support with proper focus management
- ARIA compliance- Use proper ARIA attributes for screen readers
- Visual accessibility- Maintain high contrast and clear focus indicators
- Motion preferences- Consider users who prefer reduced motion

Performance

- Respect motion preferences- Consider users who prefer reduced motion
- Optimize animations- Use CSS transforms and opacity for smooth performance
- Batch state changes- Avoid rapid successive state changes
- Use appropriate variants- Choose variants based on performance requirements

RADIO-BUTTON

The

```
<aava-radio-button>
```

component provides a radio button group for single-choice selections with multiple layout orientations, size options, custom color theming, and smooth interaction effects. It includes full Angular forms integration and accessibility compliance.

How to use

```
import {  
    AavaRadioButtonComponent  
} from "@aava/play-core";  
;
```

Basic Usage

Simple radio button group implementation with multiple options and two-way data binding.

```
<aava-radio-button  
[options]="[  
    { label: 'Option 1', value: 'option1' },  
    { label: 'Option 2', value: 'option2' },  
    { label: 'Option 3', value: 'option3' },  
]>  
name="basic-radio"  
selectedValue="option1"  
(selectedValueChange)="onRadioChange($event)"  
>  
</aava-radio-button>  
---  
onRadioChange(value: string) {
```

```
        console.log('Radio button selection changed:', value);
    }
```

Orientations

The radio button component supports both horizontal and vertical layouts to accommodate different design requirements.

```
<aava-radio-button

[options]="[

{ label: 'Option 1', value: 'option1' },
{ label: 'Option 2', value: 'option2' },
{ label: 'Option 3', value: 'option3' },

]"

name="vertical-radio"
orientation="vertical"
selectedValue="option2"
(selectedValueChange)="onRadioChange('vertical', $event)"

>
</aava-radio-button>

<aava-radio-button

[options]="[

{ label: 'Option 1', value: 'option1' },
{ label: 'Option 2', value: 'option2' },
{ label: 'Option 3', value: 'option3' },

]"

name="horizontal-radio"
orientation="horizontal"
selectedValue="option3"
(selectedValueChange)="onRadioChange('horizontal', $event)"

>
</aava-radio-button>
```

```
onRadioChange(orientation: string, value: string) {  
  console.log(`[${orientation}] orientation selection changed:`, value);  
}
```

Available Orientations

- vertical- Stacked layout with options arranged vertically (default)
- horizontal- Inline layout with options arranged horizontally

Sizes

Three size options to match different interface densities and visual hierarchies.

```
<aava-radio-button
```

```
[options]="[  
  { label: 'Option 1', value: 'option1' },  
  { label: 'Option 2', value: 'option2' },  
  { label: 'Option 3', value: 'option3' }  
]"
```

```
name="small-radio"
```

```
size="sm"
```

```
selectedValue="option1"
```

```
>
```

```
</aava-radio-button>
```

```
<aava-radio-button
```

```
[options]="[  
  { label: 'Option 1', value: 'option1' },  
  { label: 'Option 2', value: 'option2' },  
  { label: 'Option 3', value: 'option3' }  
]"
```

```
name="medium-radio"
```

```
size="md"
```

```
selectedValue="option2"
>
</aava-radio-button>

<aava-radio-button
[options]="[

{ label: 'Option 1', value: 'option1' },
{ label: 'Option 2', value: 'option2' },
{ label: 'Option 3', value: 'option3' }

]"
name="large-radio"
size="lg"
selectedValue="option3"
>
</aava-radio-button>
```

Available Sizes

- sm (Small)- Compact size for dense interfaces
- md (Medium)- Standard size for most use cases (default)
- lg (Large)- Prominent size for important selections

States

Various states for different user scenarios including selection, disabled, and interactive states.

```
<aava-radio-button
[options]="[

{ label: 'Option 1', value: 'option1' },
{ label: 'Option 2', value: 'option2' },
{ label: 'Option 3', value: 'option3' },

]"
name="default-states"
selectedValue=""
```

```

>
</aava-radio-button>
<aava-radio-button
[options]="[

{ label: 'Option 1 (Disabled)', value: 'option1', disabled: true },
{ label: 'Option 2', value: 'option2' },
{ label: 'Option 3', value: 'option3' }

]"
name="disabled-states"
selectedValue="option2"
>
</aava-radio-button>
<aava-radio-button
[options]="[

{ label: 'Option 1 (Disabled)', value: 'option1', disabled: true },
{ label: 'Option 2 (Disabled)', value: 'option2', disabled: true },
{ label: 'Option 3 (Disabled)', value: 'option3', disabled: true },

]"
name="all-disabled"
selectedValue="option3"
>
</aava-radio-button>

```

Available States

- Unselected- Default state for non-selected options
- Selected- Active state showing the selected option
- Disabled- Non-interactive state for unavailable options

Custom Colors

Comprehensive color customization for radio buttons and labels to match design requirements.

```

<aava-radio-button
[options]="[
{ label: 'Option 1', value: 'option1' },
{ label: 'Option 2', value: 'option2' },
{ label: 'Option 3', value: 'option3' },
]"
name="full-custom"
color="#8e44ad"
labelColor="#1996FC"
selectedValue="option2"
>
</aava-radio-button>

```

Color Customization Options

- labelColor- Custom color for option labels
- color- Cutom color for the dot

Form Integration

Form integration with ControlValueAccessor implementation for reactive from

```

<div class="demo-section center-demo">
<div class="forms-grid">
<h4>Reactive Form</h4>
<form [formGroup]="reactiveForm" (ngSubmit)="onReactiveSubmit()">
<div class="form-group">
<div class="form-field">
<label>Theme Selection:</label>
<aava-radio-button
formControlName="theme"
[options]="themeOptions"
name="theme-selection">

```

```
(selectedValueChange)="onReactiveThemeChange($event)"

></aava-radio-button>

</div>

<div class="form-field">

<label>Size Preference:</label>

<aava-radio-button

formControlName="size"

[options]="sizeOptions"

name="size-selection"

(selectedValueChange)="onReactiveSizeChange($event)"

></aava-radio-button>

</div>

</div>

<aava-button

label="Submit Form"

variant="primary"

[disabled]={!reactiveForm.valid"

>

</aava-button>

<div class="form-status">

<p><strong>Form Status:</strong> {{ reactiveForm.status }}</p>

</div>

<div *ngIf="showFormvalue">

<span>Form Value</span>

<pre>

{{ reactiveForm.value | json }}

</pre>

>

</div>
```

```

</form>
</div>
</div>
---
reactiveForm: FormGroup;
showFormvalue = false;
themeOptions: RadioOption[] = [
{ label: 'Light Theme', value: 'light' },
{ label: 'Dark Theme', value: 'dark' },
{ label: 'Auto Theme', value: 'auto' },
];
sizeOptions: RadioOption[] = [
{ label: 'Small', value: 'small' },
{ label: 'Medium', value: 'medium' },
{ label: 'Large', value: 'large' },
];
constructor(private fb: FormBuilder) {
this.reactiveForm = this.fb.group({
theme: ['', Validators.required],
size: ['medium', Validators.required],
});
}
onReactiveSubmit() {
if (this.reactiveForm.valid) {
this.showFormvalue = true;
console.log('Reactive form submitted:', this.reactiveForm.value);
}
}

```

Reactive Form Features

- FormControl binding- Direct integration with Angular reactive forms
- Value tracking- Proper change detection and emission
- Disabled control- Programmatic enable/disable support

Accessibility Support

The Radio Button component provides comprehensive accessibility features to ensure compatibility with screen readers, keyboard navigation, and mobile devices.

ARIA Support

- role="radiogroup"- Container identifies as a radio button group
- role="radio"- Each option is properly identified as a radio button
- aria-checked- Indicates selection state (true/false)
- aria-disabled- Identifies disabled options
- aria-label- Describes the radio group ("Radio button group with X options")
- aria-labelledby- Links radio buttons to their text labels
- aria-describedby- Provides additional context for screen readers
- aria-hidden="true"- Hides decorative elements from assistive technology

Keyboard Navigation

- Arrow Keys ($\uparrow\downarrow\leftarrow\rightarrow$)- Navigate between radio options
- Space/Enter- Select the currently focused option
- Tab- Enter and exit the radio group
- Circular Navigation- Arrow keys wrap around when reaching first/last option
- Skip Disabled- Navigation automatically skips disabled options
- Visual Focus- Clear focus indicators show current keyboard position

WCAG 2.1 Compliance

- Level AA compliant for color contrast
- Keyboard Accessible- All functionality available via keyboard
- Focus Management- Logical focus order and clear indicators
- Semantic Structure- Proper HTML roles and relationships
- Error Prevention- Clear state indication prevents user confusion

API Reference

Inputs

[TABLE]

Property	Type	Description
options	RadioOption[] []	Array of radio button options
name	string ''	HTML name attribute for the radio group
selectedValue	string ''	Currently selected option value
size	'sm' 'md' 'lg' 'md'	Size of radio buttons
orientation	'horizontal' 'vertical' 'vertical'	Layout orientation
color	string ''	Custom color for radio button and glow
labelColor	string ''	Custom color for option labels
radio	'dot' 'none' 'dot'	Whether to show the inner dot
animation	'none' 'shadow' 'none'	Animation effect for selection

[/TABLE]

Outputs

[TABLE]

Event	Type	Description
selectedValueChange	EventEmitter<string>	Emitted when selection changes

[/TABLE]

CSS Custom Properties

[TABLE]

Property	Description
--radio-group-gap	Gap between radio button options
--radio-checkmark-background	Background color of the radio button circle
--radio-checkmark-border	Border color of the radio button
--radio-checkmark-border-radius	Border radius for the radio button
--radio-checkmark-background-disabled	Background when disabled
--radio-checkmark-border-disabled	Border color when disabled

--radio-dot-background | Background color of the inner dot
--radio-dot-border-radius | Border radius of the inner dot
--radio-dot-background-disabled | Dot background when disabled
--radio-label-color | Text color for option labels
--radio-label-font | Font properties for labels
--radio-label-margin-left | Left margin for labels
--radio-label-color-disabled | Label text color when disabled
--radio-label-cursor-disabled | Cursor style when disabled
--radio-cursor | Cursor style for interactive elements
--radio-cursor-disabled | Cursor style when disabled
--radio-custom-glow-color | Custom glow color for selected state
--radio-size-sm | Size dimensions for small variant
--radio-size-md | Size dimensions for medium variant
--radio-size-lg | Size dimensions for large variant
--radio-size-sm-dot | Dot size for small variant
--radio-size-md-dot | Dot size for medium variant
--radio-size-lg-dot | Dot size for large variant
--radio-size-sm-label | Label font size for small variant
--radio-size-md-label | Label font size for medium variant
--radio-size-lg-label | Label font size for large variant

[/TABLE]

Best Practices

Design Guidelines

- Provide clear options- Use descriptive labels that clearly distinguish choices
- Limit option count- Keep radio groups to 7 or fewer options for usability
- Choose appropriate orientation- Use vertical for longer lists, horizontal for 2-4 items
- Include default selection- Pre-select the most common or safe option when appropriate
- Group related options- Only use radio buttons for mutually exclusive choices

- Consider disabled states- Disable options that aren't currently available

Performance

- Optimize option rendering- Use trackBy functions for dynamic option lists
- Debounce rapid changes- Consider debouncing for real-time validation
- Minimize re-renders- Use OnPush change detection strategy
- Efficient event handling- Use event delegation for large option lists

Form Integration

- Test with forms- Ensure proper integration with your form validation
- Handle validation- Provide clear feedback for required selections
- Consider reset behavior- Define what happens when forms are reset
- Support reactive forms- Use proper ControlValueAccessor implementation

TOGGLE

A sleek and intuitive switch component for toggling between binary states (on/off, enabled/disabled). Features smooth animations, multiple sizes, flexible label positioning, and comprehensive accessibility support for enhanced user experience.

How to use

```
import
{
  AavaToggleComponent
}
from
"@aava/play-core"
;
```

Basic Usage

Simple toggle implementations with labels and basic functionality.

[Error extracting code]

Sizes

Four size variants to accommodate different interface requirements and visual hierarchy.

[Error extracting code]

Available Sizes

- xs (Extra Small)- Extra compact size for very dense interfaces
- sm (Small): Compact for dense interfaces and secondary controls
- md (Medium): Standard size for most use cases (default)
- lg (Large): Prominent for primary settings and better accessibility

Positions

Flexible label positioning for optimal layout integration.

[Error extracting code]

Position Options

- Left: Label appears to the left of the switch (default)

- Right: Label appears to the right of the switch

States

Different toggle states including checked, unchecked, and disabled variations.

[Error extracting code]

State Management

- Checked: Active/enabled state with visual feedback
- Unchecked: Inactive/disabled state
- Disabled: Non-interactive state with reduced opacity
- Focus: Keyboard navigation with clear focus indicators

Animation

[Error extracting code]

Events

Event handling for user interactions and state changes.

[Error extracting code]

Event Features

- State Change: Emits boolean value on toggle
- Keyboard Support: Space and Enter key activation
- Click Handling: Mouse and touch interaction support
- Disabled Prevention: No events when disabled

Thumb Icon

Add icons to the toggle thumb to visually indicate checked and unchecked states. This is useful for enhancing clarity, especially in settings or preference toggles.

[Error extracting code]

Features

- Show different icons for checked and unchecked states
- Fully customizable icon names (uses Lucide icon set)
- Works with all sizes and positions

- Accessible and keyboard-friendly

Accessibility

Built-in accessibility features ensuring inclusive user experience.

[Error extracting code]

Accessibility Features

- Keyboard Navigation: Tab navigation and Space/Enter activation
- ARIA Support: Proper switch role and state announcements
- Screen Reader: Descriptive labels and state information
- Focus Management: Clear visual focus indicators
- High Contrast: Enhanced borders and outlines for visibility
- Motion Preferences: Respects reduced motion accessibility settings

API Reference

Inputs

[TABLE]

Property | Type | Default | Description

size | 'xs' | 'sm' | 'md' | 'lg' | 'md' | Size variant of the toggle switch

title | string | '' | Label text displayed next to the toggle

position | 'left' | 'right' | 'left' | Position of the label relative to switch

disabled | boolean | false | Whether the toggle is disabled

checked | boolean | false | Current checked state of the toggle

animation | boolean | true | Whether to enable smooth animations

showIcons | boolean | false | Show icons in the thumb for checked/unchecked

checkedIcon | string | 'check' | Icon name for the checked state (Lucide icon)

uncheckedIcon | string | 'x' | Icon name for the unchecked state (Lucide icon)

[/TABLE]

Outputs

[TABLE]

Event | Type | Description

checkedChange | EventEmitter<boolean> | Emitted when toggle state changes

[/TABLE]

Methods

[TABLE]

Method | Parameters | Return Type | Description

onToggle() | - | void | Toggle the checked state (if not disabled)

onKeyDown() | event: KeyboardEvent | void | Handle keyboard activation (Space/Enter)

[/TABLE]

Properties

[TABLE]

Property | Type | Description

titleName | string | null | Generated ID for label association

[/TABLE]

CSS Custom Properties

[TABLE]

Property | Description

--toggle-icon-checked-color | Color of the icon when toggle is checked

--toggle-icon-unchecked-color | Color of the icon when toggle is unchecked

--text-color-secondary | Secondary text color used in toggle

--toggle-gap | Gap between toggle and label

--toggle-label-text | Color of the label text

--toggle-font-size-xs | Font size for extra small label

--toggle-font-size-sm | Font size for small label

--toggle-font-size-md | Font size for medium label

--toggle-font-size-lg | Font size for large label

--toggle-label-disabled-text | Color of label text when toggle is disabled

--toggle-border-radius | Border radius of toggle elements

--toggle-track-background | Background color of the toggle track

--toggle-track-checked-background-gradient | Background gradient for checked state
--toggle-size-xs-width | Width of extra small toggle switch
--toggle-size-xs-height | Height of extra small toggle switch
--toggle-size-xs-thumb-size | Size of the thumb for extra small toggle
--toggle-size-sm-width | Width of small toggle switch
--toggle-size-sm-height | Height of small toggle switch
--toggle-size-sm-thumb-size | Size of the thumb for small toggle
--toggle-size-md-width | Width of medium toggle switch
--toggle-size-md-height | Height of medium toggle switch
--toggle-size-md-thumb-size | Size of the thumb for medium toggle
--toggle-size-lg-width | Width of large toggle switch
--toggle-size-lg-height | Height of large toggle switch
--toggle-size-lg-thumb-size | Size of the thumb for large toggle
--toggle-glass-shadow | Glass shadow effect for toggle
--toggle-thumb-background | Background color of the toggle thumb

[/TABLE]

Best Practices

Design Guidelines

- Use toggles for immediate binary actions that take effect instantly
- Place labels on the left for left-to-right reading patterns
- Choose appropriate sizes based on interface hierarchy
- Use disabled state for features unavailable to current user
- Consider animation preferences for better accessibility

Accessibility

- Always provide meaningful titles that describe the toggle purpose
- Use proper ARIA attributes for screen reader compatibility
- Ensure sufficient color contrast for all states
- Test keyboard navigation thoroughly

- Consider reduced motion preferences

Performance

- Avoid frequent programmatic state changes that trigger animations
- Use OnPush change detection strategy for optimal performance
- Consider disabling animations in performance-critical scenarios

DROPODOWN

AUTOCOMPLETE

DATE-PICKER

The

```
<aava-calendar>
```

component provides a robust, accessible, and highly customizable calendar interface for selecting single dates or date ranges. It features structured input fields, keyboard navigation, glassmorphism effects, and seamless integration with Angular forms. The component is designed for both standalone display and as a building block for date pickers and scheduling interfaces.

How to use

```
import {  
  AavaCalendarComponent  
}  
from "@aava/play-core"  
;
```

Basic Usage

Simple calendar for single date selection with structured input fields.

```
<aava-datepicker (dateSelected)="onDateSelected($event)"> </aava-datepicker>
```

```
onDateSelected(date: Date) {  
  console.log('Selected date:', date);  
}
```

Range Selection

Enable range mode for selecting a start and end date with dual structured inputs.

```
<aava-datepicker [isRange]="true" (rangeSelected)="onRangeSelected($event)">  
</aava-datepicker>
```

```
selectedRange: DateRange = { start: null, end: null };  
onRangeSelected(range: DateRange) {
```

```
this.selectedRange = range;  
console.log('Selected range:', range);  
}
```

Always Open / Embedded Mode

Display the calendar inline (always open) for dashboards and embedded views.

```
<aava-datepicker [alwaysOpen]="true" (dateSelected)="onDateSelected($event)">  
</aava-datepicker>
```

```
selectedDate: Date | null = null;  
onDateSelected(date: Date) {  
  this.selectedDate = date;  
  console.log('Selected date:', date);  
}
```

Customization Options

Showcase selector shape and various customization options.

```
<aava-datepicker  
[alwaysOpen]="true"  
selectorShape="square"  
(dateSelected)="onDateSelected($event)">  
</aava-datepicker>  
<aava-datepicker  
[alwaysOpen]="true"  
selectorShape="circle"  
(dateSelected)="onDateSelected($event)">  
</aava-datepicker>
```

```
onDateSelected(date: Date) {  
  console.log('Selected date:', date);  
}
```

Size Variants

Five size variants to accommodate different design requirements and space constraints.

```
<aava-datepicker  
(dateSelected)="onDateSelected($event)"  
calendarSize="xs"  
label="Extra Small"  
>  
</aava-datepicker>  
<aava-datepicker  
(dateSelected)="onDateSelected($event)"  
calendarSize="sm"  
label="Small"  
>  
</aava-datepicker>  
<aava-datepicker  
(dateSelected)="onDateSelected($event)"  
calendarSize="md"  
label="Medium"  
>  
</aava-datepicker>  
<aava-datepicker  
(dateSelected)="onDateSelected($event)"  
calendarSize="lg"  
label="Large"  
>  
</aava-datepicker>
```

```
<aava-datepicker  
(dateSelected)="onDateSelected($event)"  
calendarSize="xl"  
label="Extra Large"  
>  
</aava-datepicker>  
---  
onDateSelected(date: Date) {  
  console.log('Selected date:', date);  
}
```

Available Sizes

- xs (Extra Small)- Compact size for tight spaces and dense layouts
- sm (Small)- Smaller than default, suitable for secondary interfaces
- md (Medium)- Default size, balanced for most use cases
- lg (Large)- Larger size for enhanced readability and touch interfaces
- xl (Extra Large)- Maximum size for accessibility and prominent displays

Size Characteristics

Each size variant affects:

- Input field dimensions- Height, padding, and font sizes
- Calendar popup sizing- Overall dimensions and spacing
- Day selector sizes- Individual day cell dimensions
- Navigation elements- Button sizes and spacing
- Typography scaling- Font sizes for headers and content

Accessibility Features

Built-in accessibility features ensuring WCAG compliance and inclusive user experience.

[Error extracting code]

Features

- Single date and range selection with visual range indicators
- Structured input fields for direct date entry (DD/MM/YYYY format)
- Multiple size variants (xs, sm, md, lg, xl) for flexible layouts
- Keyboard navigation (arrow keys, Enter, Tab, Escape)
- Month/year navigation with dropdown selectors
- Customizable selector shape (square or circle)
- Glassmorphism surface effects with multiple strength levels
- Weekday format options (1, 2, or 3 letter formats)
- Full accessibility (ARIA, focus management, screen reader support)
- Angular forms integration (ControlValueAccessor implementation)
- Auto-advance input segments for seamless data entry
- Input validation and error handling
- Responsive design with mobile-friendly interactions

API Reference

Inputs

[TABLE]

Property	Type	Default	Description
isRange	boolean	false	Enable range selection mode
selectedDate	Date	null	Selected date (single mode)
dateRange	{ start: Date null, end: Date null }	{ start: null, end: null }	Selected date range (range mode)
weekdayFormat	1 2 3 3	Weekday label format: 1 = single letter, 2 = two-letter, 3 = three-letter	
alwaysOpen	boolean	false	Display calendar inline (always open)
selectorShape	'square' 'circle'	'square'	Shape of day selector
calendarSize	'xs' 'sm' 'md' 'lg' 'xl' 'md'	'md'	Size variant for calendar and input elements
surface	boolean	false	Enable glassmorphism surface effect
surfaceStrength	'medium' 'strong' 'max' undefined	'medium'	Glassmorphism intensity

[/TABLE]

Outputs

[TABLE]

Event | Type | Description

dateSelected | EventEmitter<Date> | Emitted when a date is selected (single mode)

rangeSelected | EventEmitter<{ start: Date, end: Date }> | Emitted when a date range is selected (range mode)

[/TABLE]

Computed Properties

[TABLE]

Property | Type | Description

weekDays | string[] | Computed weekday labels based on format

yearRange | number[] | Array of years (current ± 50 years)

calendarDays | CalendarDay[] | Array of days for current month display

[/TABLE]

Interfaces

interface

DateRange

{

start

:

Date

|

null

;

end

:

Date

|

null

```
;  
}  
  
interface  
CalendarDay  
{  
date  
:  
Date  
;  
:  
isCurrentMonth  
:  
boolean  
;  
isToday  
:  
boolean  
;  
isSelected  
:  
boolean  
;  
isInRange  
:  
boolean  
;  
isRangeStart  
:  
boolean  
;
```

```
isRangeEnd  
:  
boolean  
;  
}
```

CSS Custom Properties

[TABLE]

Property	Description
--calendar-font-family	Font family for calendar text
--calendar-size-sm-font	Small font size for calendar elements
--calendar-size-md-font	Medium font size for headers (default)
--calendar-input-padding	Padding for input fields
--calendar-input-border	Border for input fields
--calendar-input-border-radius	Border radius for input fields
--calendar-input-background	Background color for input fields
--calendar-popup-background	Background for calendar popup
--calendar-popup-border	Border for calendar popup
--calendar-popup-border-radius	Border radius for calendar popup
--calendar-popup-shadow	Shadow for calendar popup
--calendar-popup-z-index	Z-index for calendar popup
--calendar-nav-button-hover-background	Hover background for navigation buttons

[/TABLE]

Accessibility

The Calendar component is built with comprehensive accessibility features:

- Semantic HTML: Uses appropriate roles and elements for calendar, grid, and buttons
- ARIA attributes: Provides ARIA labels for navigation, days, and range selection
- Keyboard navigation: Full support for arrow keys, Enter, Tab, and Escape
- Screen reader support: Announces current month/year, selected dates, and range

- Focus management: Clear focus indicators and logical tab order
- High contrast: Supports high contrast and custom themes
- Reduced motion: Respects user motion preferences

Keyboard Shortcuts

- Arrow keys: Move focus between days
- Enter/Space: Select date or range endpoint
- Tab/Shift+Tab: Move between input segments and controls
- Escape: Close popup (if not always open)
- Home/End: Navigate to first/last day of month
- Page Up/Down: Navigate to previous/next month

Structured Input Navigation

- Tab: Move between day, month, year segments
- Arrow keys: Navigate within segments
- Auto-advance: Automatically moves to next segment when complete
- Validation: Real-time validation of date inputs

Best Practices

Usage Guidelines

- Single Date: Use for appointment booking, event scheduling, or simple date selection
- Range Selection: Use for booking periods, analytics date ranges, or vacation planning
- Always Open: Use for dashboards, embedded views, or when space allows
- Size Selection: Choose appropriate size based on context and space constraints
- Surface Effects: Use glassmorphism for modern, layered UI designs
- Forms Integration: Leverage ControlValueAccessor for seamless form integration

Size Selection Guidelines

- XS: Use in data tables, compact forms, or when space is extremely limited
- SM: Use in secondary interfaces, sidebars, or when you need subtle presence
- MD: Use as the default for most applications and primary interfaces
- LG: Use in touch interfaces, mobile applications, or when enhanced readability is needed

- XL: Use in accessibility-focused applications or when maximum visibility is required

Implementation Considerations

- Input Validation: Always validate user input from structured fields
- Error Handling: Provide clear feedback for invalid dates
- Mobile Experience: Test touch interactions and responsive behavior
- Performance: Calendar renders efficiently with OnPush change detection
- Theming: Use semantic tokens for consistent design system integration

Accessibility Implementation

- Screen Reader Testing: Test with NVDA, JAWS, or VoiceOver
- Keyboard Testing: Verify complete keyboard navigation flow
- Focus Management: Ensure logical tab order and clear focus indicators
- Color Contrast: Maintain sufficient contrast ratios (4.5:1 minimum)
- Motion Preferences: Respect user's reduced motion settings

TIME-PICKER

The

```
<aava-time-picker>
```

component is an advanced time selection interface that combines a clean display mode with an interactive scroll-based picker. It features smooth scrolling animations, inline editing capabilities, keyboard navigation, and comprehensive validation for precise time input in 12-hour format.

How to use

```
import {  
    AavaTimePickerComponent  
} from "@aava/play-core";
```

Basic Usage

A simple time picker with default behavior and display mode.

```
<aava-time-picker  
    [size]="'lg'"  
    (timeSelected)="onTimeSelected($event)"></aava-time-picker>  
  
---  
selectedTime = '';  
  
onTimeSelected(time: string) {  
    this.selectedTime = time;  
    console.log('Selected time:', time);  
}
```

Sizes

Four size options to fit different layout requirements and visual hierarchies.

```
<aava-time-picker size="sm"></aava-time-picker>  
<aava-time-picker size="md"></aava-time-picker>  
<aava-time-picker size="lg"></aava-time-picker>  
<aava-time-picker size="xl"></aava-time-picker>
```

Available Sizes

- sm (Small)- Compact size for dense interfaces
- md (Medium)- Standard size for most use cases (default)
- lg (Large)- Prominent size for primary actions
- xl (Extra Large)- Extra large size for hero sections and CTAs

Scroll Mode Interface

Interactive scroll-based time selection with smooth animations and visual feedback.

```
<aava-time-picker></aava-time-picker>  
---  
selectedTime = '';  
scrollEvents: string[] = [];  
onTimeSelected(time: string) {  
    this.selectedTime = time;  
    console.log('Selected time:', time);  
}  
onScrollEvent(event: any) {  
    this.scrollEvents.unshift(  
        `Scroll event: ${new Date().toLocaleTimeString()}`  
    );  
    if (this.scrollEvents.length > 5) {  
        this.scrollEvents.pop();  
    }  
}
```

Keyboard Navigation

Full keyboard accessibility with arrow keys, number input, and focus management.

```
<aava-time-picker></aava-time-picker>

---

selectedTime = "";

keyboardEvents: string[] = [];

onTimeSelected(time: string) {

this.selectedTime = time;

console.log('Selected time:', time);

}

onKeyboardEvent(event: string) {

this.keyboardEvents.unshift(

`Keyboard: ${event} - ${new Date().toLocaleTimeString()}`

);

if (this.keyboardEvents.length > 5) {

this.keyboardEvents.pop();

}

}

}
```

Validation and Constraints

Comprehensive validation with proper error handling and boundary enforcement.

```
<aava-time-picker (timeSelected)="onTimeSelected($event)"></aava-time-picker>

<aava-time-picker

[customStyles]="errorStyles"

(timeSelected)="onTimeSelected($event)"

>

</aava-time-picker>

---

selectedTime = "";

errorStyles = {

'--timepicker-background': '#fef2f2',
```

```
'--timepicker-border': '2px solid #ef4444',
'--timepicker-display-text': '#dc2626',
'--timepicker-icon-color': '#ef4444',
};

onTimeSelected(time: string) {
  this.selectedTime = time;
  console.log('Selected time:', time);
}
```

Custom Styling

Customizable appearance with CSS custom properties and theme integration.

```
<aava-time-picker [customStyles]="darkStyles"> </aava-time-picker>

<aava-time-picker [customStyles]="blueStyles"> </aava-time-picker>

<aava-time-picker [customStyles]="greenStyles"> </aava-time-picker>

---

selectedTime = "";

darkStyles = {

  '--timepicker-background': '#374151',
  '--timepicker-border': '1px solid #4b5563',
  '--timepicker-display-text': '#f9fafb',
  '--timepicker-icon-color': '#f9fafb',
  '--timepicker-scroll-background': '#1f2937',
  '--timepicker-time-item-text': '#d1d5db',
  '--timepicker-time-item-selected-text': '#60a5fa',
};

blueStyles = {

  '--timepicker-background': '#dbeafe',
  '--timepicker-border': '1px solid #3b82f6',
  '--timepicker-display-text': '#1e40af',
```

```

'--timepicker-icon-color': '#3b82f6',
'--timepicker-scroll-background': '#bfdbfe',
'--timepicker-time-item-text': '#1e3a8a',
'--timepicker-time-item-selected-text': '#1d4ed8',
};

greenStyles = {
'--timepicker-background': '#dcfce7',
'--timepicker-border': '1px solid #22c55e',
'--timepicker-display-text': '#15803d',
'--timepicker-icon-color': '#22c55e',
'--timepicker-scroll-background': '#bbf7d0',
'--timepicker-time-item-text': '#166534',
'--timepicker-time-item-selected-text': '#16a34a',
};

onTimeSelected(time: string) {
  this.selectedTime = time;
  console.log('Selected time:', time);
}

```

Features

Dual Interface Modes

- Display Mode: Clean, compact time display with click-to-edit functionality
- Scroll Mode: Interactive scroll-based selection with smooth animations
- Seamless Transition: Automatic switching between modes based on user interaction

Scroll-Based Selection

- Smooth Scrolling: Hardware-accelerated scroll animations with momentum
- Visual Feedback: Clear selection indicators and hover states
- Boundary Enforcement: Prevents scrolling beyond valid time ranges
- Padding System: Intelligent padding for optimal centering and selection

Inline Editing

- Direct Input: Click any time item to edit directly in place
- Real-time Validation: Immediate validation and formatting
- Keyboard Support: Full keyboard navigation within inline inputs
- Auto-formatting: Automatic formatting and validation on blur

Advanced Interactions

- Mouse Wheel Support: Scroll through time values with mouse wheel
- Touch Support: Optimized for touch devices and mobile interaction
- Focus Management: Proper focus trapping and restoration
- Event Handling: Comprehensive event management and propagation control

Accessibility Features

- Keyboard Navigation: Full keyboard accessibility with arrow keys
- Screen Reader Support: Proper ARIA attributes and semantic structure
- Focus Indicators: Clear visual focus indicators
- Reduced Motion: Respects user motion preferences

API Reference

Inputs

[TABLE]

Property | Type | Description

size | TimePickerSize | Size variant | 'sm' | 'md' | 'lg' | 'xl' | for the time picker (default: | 'md' |)

[/TABLE]

Outputs

[TABLE]

Event | Type | Description

timeSelected | EventEmitter<string> | Emitted when a time value is selected or changed

[/TABLE]

Properties

[TABLE]

Property | Type | Default | Description

hours | string | "" | Current hours value (01-12)

minutes | string | "" | Current minutes value (00-59)

period | string | "" | Current period value (AM/PM)

displayTime | string | "" | Formatted display time string

isFocused | boolean | false | Whether the time picker is in focused state

[/TABLE]

ViewChild References

[TABLE]

Reference | Type | Description

hoursScroll | ElementRef | Reference to hours scroll container

minutesScroll | ElementRef | Reference to minutes scroll container

periodScroll | ElementRef | Reference to period scroll container

[/TABLE]

Internal Properties

[TABLE]

Property | Type | Description

showInlineInput | boolean | Whether inline input is currently visible

inlineInputType | 'hours' | 'minutes' | 'period' | null | Type of current inline input

inlineInputValue | string | Current value of inline input

inlineInputPosition | object | Position and dimensions of inline input

clickedItemValue | string | Value of clicked item for inline editing

centeredHour | string | Currently centered hour in scroll view

centeredMinute | string | Currently centered minute in scroll view

centeredPeriod | string | Currently centered period in scroll view

[/TABLE]

Readonly Arrays

[TABLE]

Property | Type | Description

hoursList | string[] | Array of hours (01-12) with padding

minutesList | string[] | Array of minutes (00-59) with padding

periodList | string[] | Array of periods (AM, PM) with padding

[/TABLE]

Methods

Public Methods

[TABLE]

Method | Parameters | Return Type | Description

onDisplayClick() | Event | void | Handles display area click to enter scroll mode

onIconClick() | Event | void | Handles clock icon click to enter scroll mode

onTimeItemClick() | Event, type, value | void | Handles time item click for selection/editing

onScrollEvent() | Event, type | void | Handles scroll events for time selection

centerScrollToValue() | type, value | void | Centers scroll to specific time value

updateDisplayTime() | None | void | Updates the formatted display time

emitTimeSelected() | None | void | Emits the timeSelected event

autoSelectCurrentScrollValues() | None | void | Forces selection of currently visible values

[/TABLE]

Private Methods

[TABLE]

Method | Parameters | Return Type | Description

generatePaddedList() | string[] | string[] | Generates padded list for smooth scrolling

enforceScrollBoundaries() | type | void | Enforces scroll boundaries for each column

handleWheelEvent() | WheelEvent, type | void | Handles mouse wheel events

updateSelectionForColumn() | type | void | Updates selection for specific column

getFormattedTime() | None | string | Returns formatted time string

onDocumentClick() | MouseEvent | void | Handles document clicks for focus management

onInlineInputChange() | Event | void | Handles inline input value changes

onInlineInputKeyPress() | KeyboardEvent | void | Handles inline input keyboard events
onInlineInputBlur() | None | void | Handles inline input blur events
applyInlineInput() | None | void | Applies inline input value with validation
validateInlineHours() | string | string | Validates and formats hours input
validateInlineMinutes() | string | string | Validates and formats minutes input
validateInlinePeriod() | string | string | Validates and formats period input
closeInlineInput() | None | void | Closes inline input and restores normal view

[/TABLE]

CSS Classes

The component provides several CSS classes for styling:

[TABLE]

Class Name	Description
.time-picker-container	Main container wrapper
.time-picker-input	Input container with border and styling
.time-display	Display mode time text
.time-scroll-container	Scroll mode container
.inline-input-overlay	Inline input overlay container
.inline-scroll-input	Inline input field
.separator	Time separator (:)
.time-scroll-column	Individual scroll column container
.period-column	Period column with smaller width
.scroll-area	Scrollable area container
.time-item	Individual time item
.selected	Selected time item styling
.padding-item	Padding item (invisible)
.hidden-item	Hidden item during inline editing
.icon-wrapper	Clock icon container

[/TABLE]

CSS Custom Properties

[TABLE]

Property	Description
--timepicker-background	Background color of the time picker
--timepicker-border	Border of the time picker
--timepicker-border-radius	Border radius of the time picker
--timepicker-shadow	Shadow for the time picker container
--timepicker-input-height	Height of the input field
--timepicker-input-min-width	Minimum width of the input field
--timepicker-input-padding-horizontal	Horizontal padding inside input
--timepicker-input-background	Background color for the input
--timepicker-input-border-width	Border width of the input
--timepicker-input-border-color	Border color of the input
--timepicker-input-border-radius-small	Border radius for small input variant
--timepicker-input-padding-small	Padding for small input variant
--timepicker-input-shadow-overlay	Overlay shadow for input field
--timepicker-input-text	Text color for input
--timepicker-display-font-family	Font family for display text
--timepicker-display-text	Text color for display mode
--timepicker-text-to-icon-gap	Gap between display text and icon
--timepicker-scroll-background	Background color for scroll container
--timepicker-scroll-gap-hours-minutes	Gap between hours and minutes in scroll
--timepicker-time-item-height	Height of individual time items
--timepicker-time-item-background	Background color for time items
--timepicker-time-item-padding	Padding for time items
--timepicker-time-item-border-radius	Border radius for time items
--timepicker-time-item-transition	Transition timing for time items
--timepicker-time-item-text	Text color for time items
--timepicker-time-item-hover-background	Background on hover for time items

--timepicker-time-item-hover-text | Text color on hover for time items
--timepicker-time-item-selected-text | Text color for selected time item
--timepicker-time-item-selected-font-weight | Font weight for selected time item
--timepicker-icon-size | Size of the time picker icon
--timepicker-icon-border-radius | Border radius for the icon
--timepicker-icon-padding | Padding inside the icon
--timepicker-icon-color | Icon color
--color-text-primary | Text color used in the separator
--timepicker-size-xsm-height | Height for extra small size variant
--timepicker-size-xsm-min-width | Minimum width for extra small size
--timepicker-size-xsm-padding-horizontal | Horizontal padding for extra small size
--timepicker-border-radius-xs | Border radius for extra small variant
--timepicker-size-xsm-font | Font for extra small size
--timepicker-size-xsm-item-height | Item height for extra small variant
--timepicker-size-xsm-icon-size | Icon size for extra small variant
--timepicker-size-sm-height | Height for small size variant
--timepicker-size-sm-min-width | Minimum width for small size
--timepicker-size-sm-padding-horizontal | Horizontal padding for small size
--timepicker-border-radius-sm | Border radius for small variant
--timepicker-size-sm-font | Font for small size
--timepicker-size-sm-item-height | Item height for small variant
--timepicker-size-sm-icon-size | Icon size for small variant
--timepicker-size-md-height | Height for medium size variant
--timepicker-size-md-min-width | Minimum width for medium size
--timepicker-size-md-padding-horizontal | Horizontal padding for medium size
--timepicker-border-radius-md | Border radius for medium variant
--timepicker-size-md-font | Font for medium size
--timepicker-size-md-item-height | Item height for medium variant
--timepicker-size-md-icon-size | Icon size for medium variant

--timepicker-size-lg-height | Height for large size variant
--timepicker-size-lg-min-width | Minimum width for large size
--timepicker-size-lg-padding-horizontal | Horizontal padding for large size
--timepicker-border-radius-lg | Border radius for large variant
--timepicker-size-lg-font | Font for large size
--timepicker-size-lg-item-height | Item height for large variant
--timepicker-size-lg-icon-size | Icon size for large variant
--timepicker-size-xlg-height | Height for extra large size variant
--timepicker-size-xlg-min-width | Minimum width for extra large size
--timepicker-size-xlg-padding-horizontal | Horizontal padding for extra large size
--timepicker-border-radius-xl | Border radius for extra large variant
--timepicker-size-xlg-font | Font for extra large size
--timepicker-size-xlg-item-height | Item height for extra large variant
--timepicker-size-xlg-icon-size | Icon size for extra large variant

[/TABLE]

Best Practices

User Experience

- Default Values: Set sensible defaults (e.g., current time) for better UX
- Visual Feedback: Ensure clear selection indicators and hover states
- Smooth Transitions: Maintain smooth animations between modes
- Responsive Design: Ensure the component works well on all screen sizes
- Loading States: Handle loading states gracefully

Performance

- Scroll Optimization: Use hardware acceleration for smooth scrolling
- Event Debouncing: Debounce scroll events to prevent performance issues
- Memory Management: Clean up event listeners and timeouts properly
- Change Detection: Use OnPush strategy for better performance
- Lazy Loading: Load time lists only when needed

Accessibility

- Keyboard Navigation: Ensure all interactions work with keyboard
- Screen Reader Support: Provide proper ARIA labels and descriptions
- Focus Management: Maintain logical focus order and trapping
- Motion Preferences: Respect user's reduced motion preferences
- High Contrast: Ensure visibility in high contrast mode

Validation

- Input Validation: Validate all user inputs thoroughly
- Boundary Enforcement: Prevent invalid time selections
- Error Handling: Provide clear error messages and recovery options
- Format Consistency: Maintain consistent time formatting
- Edge Cases: Handle edge cases like leap seconds, DST changes

Content Organization

- Clear Labels: Use descriptive labels and placeholders
- Logical Grouping: Group related time components logically
- Visual Hierarchy: Use typography and spacing for clear hierarchy
- Consistent Spacing: Maintain consistent spacing throughout
- Responsive Layout: Adapt layout for different screen sizes

Accessibility Guidelines

Screen Reader Support

- ARIA Labels: Provide descriptive labels for all interactive elements
- Live Regions: Use live regions for dynamic content updates
- State Announcements: Announce selection changes and mode switches
- Content Structure: Use semantic HTML structure for better navigation

Keyboard Navigation

- Tab Order: Ensure logical tab order through all interactive elements
- Arrow Keys: Support arrow key navigation within scroll areas
- Enter/Space: Support selection with Enter and Space keys

- Escape: Allow closing or canceling with Escape key
- Focus Indicators: Provide clear visual focus indicators

Visual Design

- High Contrast: Ensure sufficient contrast ratios for all text
- Color Independence: Don't rely solely on color for information
- Text Scaling: Support for text scaling and zoom
- Motion Sensitivity: Provide alternatives for users sensitive to motion

Input Validation

- Clear Messages: Provide clear, actionable error messages
- Real-time Feedback: Give immediate feedback for invalid inputs
- Recovery Options: Provide clear ways to correct errors
- Format Guidance: Show expected input format clearly

RANGE-SLIDER

The

Range Slider

component provides a simple and elegant way to select a numeric value or range within a defined interval.

It features smooth drag interactions, built-in accessibility, and easy integration with Angular forms for both template-driven and reactive use cases.

The

<aava-slider>

component supports multiple modes such as

single value

,

multi-range

, and

input-integrated

sliders, making it adaptable to diverse UI needs.

How to use

```
import {  
  AavaSliderComponent  
}  
from "@aava/play-core";
```

Basic Usage

The simplest version of the slider displays a single draggable handle to choose a numeric value between a default range of 0–100.

Ideal for scenarios like volume control, brightness adjustment, or progress selection.

Size Variants

The slider supports multiple size options to fit different design requirements and layouts.

Smaller sliders suit compact UIs, while medium sizes provide comfortable interaction for most use cases.

Available Size

- sm (Small): Compact slider ideal for dense layouts and mobile interfaces
- md (Medium): Standard size slider for most common use cases (default)

States

Demonstrates the slider's different states, including disabled, active, and focused.

These states help communicate interactivity and status changes to users clearly.

Multi Range Slider

The multi-range version allows selection of both minimum and maximum values, offering a more flexible range selection experience.

It's ideal for use in filters, price sliders, or any range-based data input scenarios.

Multi Range Features

- Dual Handles: Independent control of minimum and maximum values
- Range Selection: Visual indication of selected range between handles
- Collision Prevention: Handles cannot cross over each other
- Synchronized Tooltips: Both handles show their respective values

Input Type Variant

This variant combines a slider handle with a numeric input field, enabling precise manual entry alongside drag interaction.

It's especially useful in cases where exact numeric control is needed, such as filtering or budget ranges.

Input Type Features

- Dual Input Methods: Users can drag the slider or type directly in the input field
- Real-time Sync: Input field and slider stay synchronized
- Validation: Input respects min/max boundaries and step values
- Accessibility: Input field provides keyboard navigation alternative
- Responsive Design: Input field adapts to slider size variants

Icon Slider Variants

Customizable slider with icon-based thumbs for enhanced visual feedback and thematic consistency.

Icon Thumb Features

- Custom Icons: Replace default handle with Lucide icons
- Thematic Consistency: Icons that match your content context
- Multiple Variants: Various icon styles for different use cases
- Responsive Sizing: Icons scale appropriately with slider size
- Color Theming: Icons inherit slider theme colors
- Hover Effects: Enhanced visual feedback on interaction

Icon Thumb Variants

- Volume Control: Speaker/volume icons for audio controls
- Brightness: Sun/brightness icons for display settings
- Temperature: Thermometer icons for climate controls
- Speed: Gauge/speedometer icons for rate adjustments
- Rating: Star icons for rating and review systems
- Progress: Arrow or progress icons for completion tracking

Orientation

Accessibility

Built-in accessibility features ensuring WCAG compliance and inclusive user experience.

Accessibility Features

- Keyboard Navigation: Arrow keys, Home, and End key support
- ARIA Attributes: Properrole="slider",aria-valuemin,aria-valuemax,aria-valuenow
- Focus Management: Clear focus indicators and outline
- Touch Support: Optimized for touch devices
- Input Integration: Numeric input field provides alternative input method

Keyboard Controls

- Arrow Right/Up: Increase value by step amount

- Arrow Left/Down: Decrease value by step amount
- Home: Jump to minimum value
- End: Jump to maximum value

API Reference

Inputs

[TABLE]

Property	Type	Default	Description
min	number	0	Minimum value of the slider range
max	number	100	Maximum value of the slider range
value	number	0	Current value of the slider
step	number	1	Step increment for value changes
showTooltip	boolean	true	Whether to display the value tooltip
size	'sm' 'md' 'md'		Size variant of the slider
type	'default' 'input' 'default'		Display type with or without input field
multiRange	boolean	false	Enable multi-range (two-handle) slider
minValue	number	20	Minimum selected value in multi-range mode
maxValue	number	80	Maximum selected value in multi-range mode
iconStart	string	"	Icon displayed at the start of the slider track
iconEnd	string	"	Icon displayed at the end of the slider track
handleIcon	string	"	Icon displayed on the slider handle
handleIconStart	string	"	Icon displayed on the start handle (multi-range)
handleIconEnd	string	"	Icon displayed on the end handle (multi-range)
customStyles	Record<string, string>	{}	CSS custom properties override
disabled	boolean	false	Disable the slider

[/TABLE]

Outputs

[TABLE]

Event	Type	Description

valueChange | EventEmitter<number> | Emitted when the main slider value changes
minValueChange | EventEmitter<number> | Emitted when the minimum value changes
maxValueChange | EventEmitter<number> | Emitted when the maximum value changes

[/TABLE]

Methods

The component implements

ControlValueAccessor

for form integration:

[TABLE]

Method | Parameters | Description

writeValue | value: number | Set value programmatically

registerOnChange | fn: Function | Register change callback

registerOnTouched | fn: Function | Register touched callback

[/TABLE]

CSS Custom Properties

The slider supports a wide range of CSS custom properties for theming and customization:

[TABLE]

Property | Description

--slider-container-height | Height of the overall slider container

--slider-container-gap | Spacing between slider elements

--slider-input-gap | Gap between the slider track and input field

--slider-size-sm-track-height | Track height for small slider size variant

--slider-size-sm-thumb-size | Thumb size for small slider

--slider-label-font-size-sm | Label font size for small slider

--slider-label-weight-sm | Label font weight for small slider

--slider-size-md-track-height | Track height for medium slider size variant

--slider-size-md-thumb-size | Thumb size for medium slider

--slider-label-font-size-md | Label font size for medium slider

--slider-label-weight-md | Label font weight for medium slider

--slider-track-height | Height of the slider track
--slider-track-background | Background color of the slider track
--slider-track-border-radius | Border radius of the track
--slider-progress-background | Background color of the filled progress area
--slider-progress-border-radius | Border radius of the progress area
--slider-thumb-size | Size of the slider thumb
--slider-thumb-border-radius | Border radius of the thumb
--slider-thumb-inner-background | Background color inside the thumb
--slider-thumb-shadow | Shadow of the thumb
--slider-thumb-shadow-hover | Thumb shadow on hover
--slider-focus-ring | Style of the focus ring
--slider-focus-ring-offset | Offset distance of the focus ring
--slider-cursor | Cursor style when hovering over slider
--slider-tooltip-margin | Margin around the tooltip
--slider-tooltip-padding | Padding inside the tooltip
--slider-tooltip-border-radius | Border radius of the tooltip
--slider-value-color | Text color of the tooltip value
--slider-label-font-family | Font family used for labels
--slider-label-line-height | Line height for labels
--slider-mark-background | Background color of slider marks
--slider-handle-icon-width | Width of the handle icon
--slider-handle-icon-height | Height of the handle icon
--slider-input-width | Width of the input field
--slider-input-height | Height of the input field
--slider-input-padding | Padding inside the input field
--slider-input-border-radius | Border radius of the input field
--slider-input-border | Border style of the input field
--slider-input-background | Background color of the input field

--slider-input-font-size | Font size of the input text
--slider-input-font-weight | Font weight of the input text
--slider-input-font-family | Font family of the input text
--slider-input-color | Text color of the input
--slider-input-transition | Transition style for input state changes
--slider-input-focus-border-color | Border color of input when focused
--slider-input-hover-border-color | Border color of input when hovered
--slider-input-disabled-background | Background color of disabled input
--slider-input-disabled-border-color | Border color of disabled input
--slider-value-color-disabled | Text color for disabled value display
--slider-disabled-color | Color used in disabled state
--slider-disabled-rail-background | Background of the slider rail when disabled

[/TABLE]

Best Practices

Implementation Guidelines

- Use appropriate step values for your use case (1 for integers, 0.1 for decimals)
- Set meaningful min/max boundaries that make sense for your data
- Consider hiding the tooltip for inline sliders in dense layouts
- Always provide proper labels for accessibility
- Choose appropriate size variants based on your layout density
- Use input type for scenarios requiring precise numeric input

Size Selection Guidelines

- Small: Use in compact layouts, mobile interfaces, or when space is limited
- Medium: Default choice for most applications and standard layouts

Input Type Usage

- Default Type: Best for visual-only interactions and quick value selection
- Input Type: Ideal for applications requiring precise numeric input or accessibility compliance

Form Integration

- Use reactive forms for complex validation scenarios
- Implement proper error handling and validation messages
- Consider debouncing frequent value changes for performance
- Leverage input type for better form accessibility and user experience

RATING

The

```
<aava-rating>
```

component provides an intuitive and accessible star rating interface with support for half-star ratings, multiple size variants, and comprehensive keyboard navigation. Perfect for user feedback, product reviews, and any scenario requiring rating input or display.

How to use

```
import {  
    AavaRatingComponent  
} from "@aava/play-core";
```

Basic Usage

Simple rating implementation with default 5-star scale and interactive functionality.

```
<aava-rating [value]="ratingValue" (rated)="onRatingChange($event)">  
</aava-rating>
```

```
onRatingChange(value: number) {  
    console.log('Rating changed to:', value);  
}
```

Sizes

Four size variants to accommodate different interface densities and visual hierarchy requirements.

```
<!-- Different size variants -->
```

```
<aava-rating [value]="ratingValue" size="xs" (rated)="onRatingChange($event)">  
</aava-rating>  
<aava-rating [value]="ratingValue" size="sm" (rated)="onRatingChange($event)">
```

```

</aava-rating>

<aava-rating [value]="ratingValue" size="md" (rated)="onRatingChange($event)">
</aava-rating>

<aava-rating [value]="ratingValue" size="lg" (rated)="onRatingChange($event)">
</aava-rating>

<aava-rating [value]="ratingValue" [size]="40" (rated)="onRatingChange($event)">
</aava-rating>

---

ratingValue = 3.5;

ononRatingChange(value: number) {
  this.ratingValue = value;
  console.log('Rating changed to:', value);
}

```

Available Sizes

- xs (Extra Small)- 16px stars for very compact interfaces
- sm (Small)- 20px stars for dense interfaces
- md (Medium)- 24px stars for standard layouts (default)
- lg (Large)- 32px stars for prominent placements and better accessibility
- Custom- Numeric values for precise sizing requirements

Half-Star Ratings

Support for precise half-star ratings (e.g., 4.5 stars) with intuitive click positioning.

```

<!-- Half-star ratings -->

<aava-rating [value]="3.5" (rated)="onRatingChange($event)"></aava-rating>
<aava-rating [value]="4.5" (rated)="onRatingChange($event)"></aava-rating>
<aava-rating [value]="2.5" (rated)="onRatingChange($event)"></aava-rating>

<!-- Interactive half-star selection -->

<aava-rating
  [value]="currentRating"

```

```

(rated)="ratingChange($event)"
></aava-rating>
---
currentRating = 0;
onRatingChange(rating: number) {
  this.currentRating = rating;
  console.log('Rating changed to:', rating);
}

```

Half-Star Features

- Click Positioning- Left half of star = half rating, right half = full rating
- Hover Preview- Visual feedback shows potential rating before clicking
- Precise Control- Support for ratings like 3.5, 4.5, etc.
- Intuitive UX- Natural interaction pattern users expect

Readonly Mode

Display-only mode for showing existing ratings without user interaction.

```
<aava-rating [value]="4.5" [readonly]="true"></aava-rating>
```

Readonly Features

- Non-interactive- No click or hover effects
- Display Only- Perfect for showing existing ratings
- Accessibility- Maintains proper ARIA attributes
- Consistent Styling- Same visual appearance as interactive mode

Show Value

Display the numeric rating value alongside the visual stars.

```
<!-- Ratings with numeric values displayed -->
<aava-rating [value]="4.5" [showValue]="true"></aava-rating>
```

Value Display Features

- Numeric Rating- Shows exact rating (e.g., "4.5")
- Size Variants- Value text scales with star size

- Positioning- Value appears to the right of stars
- Formatting- Always shows one decimal place for precision

Custom Maximum

Flexible rating scales beyond the default 5-star system.

```
<!-- Different rating scales -->
<aava-rating
  [value]="3"
  [max]="3"
  (rated)="onRatingChange($event)"
></aava-rating>
<aava-rating
  [value]="4"
  [max]="4"
  (rated)="onRatingChange($event)"
></aava-rating>
<aava-rating
  [value]="5"
  [max]="5"
  (rated)="onRatingChange($event)"
></aava-rating>
<aava-rating
  [value]="7"
  [max]="10"
  (rated)="onRatingChange($event)"
></aava-rating>
<!-- Custom scale with half-star -->
<aava-rating
  [value]="8.5"
```

```
[max]="10"  
(rated)="onRatingChange($event)"  
></aava-rating>  
---  
onRatingChange(rating: number) {  
  console.log('Rating changed to:', rating);  
}
```

Custom Scale Features

- Flexible Range- Support for 3, 4, 5, 10, or any number of stars
- Consistent Behavior- Same interaction patterns regardless of scale
- Half-Star Support- Works with any maximum value
- Accessibility- Proper ARIA attributes for custom scales

Accessibility

Accessibility Features

- Keyboard Navigation- Full keyboard support with arrow keys
- ARIA Compliance- Proper role="radiogroup" and aria-checked attributes
- Screen Reader Support- Clear announcements of current rating
- Focus Management- Visible focus indicators for keyboard users
- High Contrast- Enhanced visibility in high contrast modes
- Motion Preferences- Respects user's reduced motion settings

Keyboard Shortcuts

- Arrow Right/Up- Increase rating by 1 star
- Arrow Left/Down- Decrease rating by 1 star
- Enter/Space- Select the currently focused star
- Tab/Shift+Tab- Navigate between stars

API Reference

Inputs

[TABLE]

Property | Type | Default | Description

value | number | 0 | Current rating value (supports halves like 4.5)

max | number | 5 | Maximum number of stars in the rating scale

readonly | boolean | false | Whether the rating is read-only (non-interactive)

size | number | 'xs' | 'sm' | 'md' | 'lg' | 'md' | Size of the stars (predefined or custom pixel values)

showValue | boolean | false | Whether to display the numeric rating value

[/TABLE]

Outputs

[TABLE]

Event | Type | Description

rated | EventEmitter<number> | Emitted when user changes the rating value

[/TABLE]

CSS Custom Properties

[TABLE]

Property | Description

--rating-label-font-family | Font family for rating label text

--rating-label-font-weight | Font weight for rating label text

--rating-label-font-size | Font size (used as line-height) for rating label text

--rating-label-color | Text color for rating label

--rating-label-letter-spacing-sm | Letter spacing for small label text

--rating-label-letter-spacing-medium | Letter spacing for medium label text

--rating-label-letter-spacing-lg | Letter spacing for large label text

--rating-value-font-size-sm | Font size for small value variants (| xs | & | sm |)

--rating-value-font-size-md | Font size for medium value variant

--rating-value-font-size-lg | Font size for large value variant

[/TABLE]

Best Practices

Design Guidelines

- Choose appropriate sizes- Use larger sizes for primary rating displays, smaller for secondary
- Consider half-star support- Enable for precise rating needs, disable for simpler interfaces
- Show value when needed- Display numeric ratings for clarity in review systems
- Use consistent scales- Stick to common scales (5-star, 10-star) for user familiarity
- Position strategically- Place ratings near relevant content for context

Accessibility

- Always provide labels- Use descriptive labels for screen reader context
- Test keyboard navigation- Ensure full keyboard accessibility
- Consider motion preferences- Respect user's reduced motion settings
- Maintain contrast- Ensure sufficient contrast for all star states
- Provide alternatives- Consider text-based rating alternatives for complex cases

Performance

- Optimize re-renders- Use OnPush change detection strategy when possible
- Efficient event handling- Optimize mouse and keyboard event handlers
- Image optimization- Use optimized SVG assets for stars
- Memory management- Clean up event listeners properly

Form Integration

- Angular Forms- Integrate with reactive and template-driven forms
- Validation- Implement appropriate validation for rating inputs
- Default values- Provide sensible defaults for new ratings
- Error handling- Handle edge cases and invalid inputs gracefully

Use Cases

- Product Reviews- E-commerce product rating systems
- Service Feedback- Customer satisfaction ratings
- Content Rating- Movie, book, or content ratings
- Skill Assessment- Employee or skill evaluation systems
- Quality Metrics- Internal quality or performance ratings

Technical Notes

Star Asset Requirements

The component expects three SVG assets:

- star-filled.svg- For fully rated stars
- star-half.svg- For half-rated stars
- star-outline.svg- For empty stars

Half-Star Logic

Half-star ratings are determined by click position:

- Left half of star =index + 0.5
- Right half of star =index + 1.0

Size Mapping

Predefined sizes map to pixel values:

- extra small: 16px
- small: 20px
- medium: 24px (default)
- large: 32px

Custom numeric values are used directly for precise sizing requirements.

Event Handling

The component handles multiple interaction types:

- Mouse: Click for selection, hover for preview
- Keyboard: Arrow navigation, Enter/Space for selection
- Touch: Click events work on touch devices
- Programmatic: Direct value changes via input binding

SEARCH-BAR

The

```
<aava-search-bar>
```

component provides a comprehensive search input solution with integrated search and send icons. Built on top of the textbox component, it offers consistent styling, multiple size variants, and flexible configuration options for various search scenarios.

How to use

```
import {  
    AavaSearchBarComponent  
} from "@aava/play-core";
```

Basic Usage

Simple search bar implementation with default settings and integrated search functionality.

```
<aava-search-bar  
    placeholder="Enter your search term..."  
    [closeButton]="true"  
    (searchClick)="onSearchClick($event)"  
    (searchChange)="onSearchChange($event)"  
    (onClose)="onClose($event)"></aava-search-bar>  
  
---  
  
onSearchClick(searchTerm: string) {  
    console.log('Search clicked with term:', searchTerm);  
}  
  
onSearchChange(event: Event) {  
    console.log('Textbox changed:', this.currentInput);
```

```
}

onClose(searchTerm: string) {
  console.log('Clear search:', searchTerm);
}

}
```

Basic Features

- Integrated Icons: Built-in search, send and close icons for intuitive user experience
- Textbox Foundation: Inherits all textbox functionality and styling
- Search Events: Emits search events when user click the send icon
- Responsive Design: Adapts to different screen sizes and container widths
- Accessibility: Full keyboard navigation and screen reader support

Variants

Simple search bar implementation with default settings and integrated search functionality.

```
<aava-search-bar closeButtonVisibility="always"></aava-search-bar>

<aava-search-bar closeButtonVisibility="hidden"></aava-search-bar>

<aava-search-bar

closeButtonVisibility="always"

[sendButton]="false"

></aava-search-bar>

<aava-search-bar [sendButton]="false"></aava-search-bar>

<aava-search-bar

placeholder="Primary"

searchIconColor="#3b82f6"

sendIconColor="#3b82f6"

></aava-search-bar>

<aava-search-bar

placeholder="Success"

searchIconColor="#10b981"

sendIconColor="#10b981"
```

```
></aava-search-bar>

<aava-search-bar
placeholder="Warning"
searchIconColor="#f59e0b"
sendIconColor="#f59e0b"
></aava-search-bar>

<aava-search-bar
placeholder="Danger"
searchIconColor="#ef4444"
sendIconColor="#ef4444"
></aava-search-bar>
```

Available Variants

- closeButtonVisibility: Controls the visibility behavior of the close (clear) button.

This input determines when the close button should be displayed based on the textbox content or user preference.

- sendButton: Controls the visibility of the send button.
- searchIconColor: Controls the search icon color
- sendIconColor: Controls the send icon color

Size

Four size variants to accommodate different interface densities and design requirements.

```
<aava-search-bar placeholder="Extra Small" size="xs"></aava-search-bar>
<aava-search-bar placeholder="Medium" size="md"></aava-search-bar>
<aava-search-bar placeholder="Large" size="lg"></aava-search-bar>
<aava-search-bar placeholder="Extra Large" size="xl"></aava-search-bar>
```

Available Sizes

- xs (Extra Small)- Compact size for minimal interfaces (16px icon)
- md (Medium)- Standard size for most search scenarios (20px icon, default)
- lg (Large)- Prominent size for important search interfaces (24px icon)
- xl (Extra Large)- Very prominent size for high-visibility search (24px icon)

Size Features

- Proportional Scaling: Icon sizes scale appropriately with component size
- Consistent Spacing: Maintains proper proportions across all size variants
- Touch Targets: All sizes meet minimum touch target requirements
- Visual Hierarchy: Larger sizes provide better emphasis for primary search areas

States

Different interaction states for various user scenarios and accessibility requirements.

```
<aava-search-bar placeholder="Default"></aava-search-bar>
```

```
<aava-search-bar placeholder="Disabled" [disabled]="true"></aava-search-bar>
```

State Features

- Default State: Normal interaction with full functionality
- Disabled State: Non-interactive state with visual feedback
- Focus State: Clear focus indicators for keyboard navigation
- Hover State: Subtle hover effects for interactive elements

API Reference

Inputs

[TABLE]

Property | Type | Default | Description

label | string? | "" | Label text for the search input

disabled | boolean | false | Whether the search bar is disabled

variant | string? | "" | Visual variant of the search bar

size | 'xs' | 'md' | 'lg' | 'xl' | 'md' | Size variant of the search bar

searchIconColor | string | '#000000' | Color for the search icon

sendIconColor | string | '#000000' | Color for the send icon

sendButton | boolean | true | Whether the send button is shown

closeButtonVisibility | 'auto' | 'always' | 'hidden' | hidden | Close button visibility

[/TABLE]

Outputs

[TABLE]

Event | Type | Description

searchClick | EventEmitter<string> | Emitted when user clicks the send icon

searchChange | EventEmitter<string> | Emitted when the search input value changes

onClose | EventEmitter<string> | Emitted when the close button is clicked

[/TABLE]

Best Practices

Design Guidelines

- Clear Visual Hierarchy: Use appropriate sizes for primary vs. secondary search areas
- Consistent Iconography: Maintain consistent icon colors across your application
- Accessible Contrast: Ensure sufficient contrast between icons and backgrounds
- Responsive Sizing: Choose sizes that work well on all target devices
- Brand Consistency: Use icon colors that align with your brand guidelines

Accessibility

- Keyboard Navigation: Ensure full keyboard accessibility for all interactions
- Screen Reader Support: Provide clear labels and descriptions for search functionality
- Focus Management: Maintain clear focus indicators for all interactive elements
- ARIA Labels: Use appropriate ARIA attributes for search context
- Color Independence: Don't rely solely on color to convey information

Performance

- Event Handling: Debounce search input events for better performance
- Icon Optimization: Use optimized SVG icons for consistent rendering
- Change Detection: Leverage OnPush strategy for optimal performance
- Memory Management: Clean up event listeners and subscriptions properly
- Bundle Size: Import only needed components to minimize bundle size

User Experience

- Clear Purpose: Make it obvious that this is a search input

- Visual Feedback: Provide immediate feedback for user interactions
- Search Suggestions: Consider adding autocomplete or search suggestions
- Loading States: Show loading indicators during search operations
- Error Handling: Provide clear error messages for failed searches

Implementation Considerations

- State Management: Properly manage search state in your application
- Search Logic: Implement efficient search algorithms and data structures
- API Integration: Handle search API calls with proper error handling
- Caching: Cache search results for better performance
- Analytics: Track search usage for user behavior insights

Search Functionality

- Real-time Search: Consider implementing search-as-you-type functionality
- Search History: Maintain search history for user convenience
- Search Filters: Add filtering options for refined search results
- Search Results: Design clear and organized search result displays
- Search Analytics: Monitor search patterns and popular queries

Technical Notes

Component Architecture

The search bar component extends the textbox component:

- Inheritance: Extends `AvaTextboxComponent` for consistent behavior
- Composition: Uses `IconComponent` for search and send icons
- Change Detection: Implements `OnPush` strategy for optimal performance
- Styling: Inherits textbox styling with search-specific customizations

Icon System

The component implements a flexible icon system:

- Dynamic Sizing: Icon sizes are computed based on component size
- Color Management: Handles disabled states with CSS variable fallbacks
- Slot System: Uses Angular slot system for icon positioning

- Interactive Elements: Send icon is clickable with proper cursor styling

Event Handling

The component manages multiple event types:

- Input Changes: Tracks search value changes in real-time
- Search Triggers: Emits search events when send icon is clicked
- Textbox Events: Inherits all textbox event handling
- Icon Interactions: Handles icon-specific interactions and styling

Size Mapping

The component maps size variants to icon dimensions:

- XS: 16px icons for compact interfaces
- MD: 20px icons for standard usage (default)
- LG: 24px icons for prominent interfaces
- XL: 24px icons for high-visibility areas

CSS Integration

The component integrates with the design system:

- CSS Variables: Uses semantic CSS variables for theming
- Responsive Design: Adapts to different screen sizes
- State Management: Handles various component states
- Accessibility: Maintains proper contrast and focus indicators

FILEUPLOAD

A comprehensive file upload component that provides multiple layout variants, drag-and-drop functionality, file validation, and preview capabilities. Built with accessibility in mind and designed for various file upload scenarios including single file uploads, multiple file selections, and enterprise file management interfaces.

How to use

```
import  
{  
  AavaFileUploadComponent  
}  
from  
"@aava/play-core"  
;
```

Note

: The FileUpload component is standalone and includes all necessary dependencies for icon, button, tag, and common modules.

Basic Usage

Simple file upload with default layout and basic functionality.

```
<aava-file-upload  
[allowedFormats]=["pdf", "doc", "docx", "txt", "jpg", "png"]  
(selectedList)="onFileChange($event)"  
>  
</aava-file-upload>  
---  
selectedFiles: File[] = [];  
onFileChange(files: File[]): void {  
  console.log('Files changed:', files);  
  this.selectedFiles = files;  
}
```

Layout Variants

```
<div>

<h1>Default Layout</h1>

<aava-file-upload

layout="default"

[allowedFormats]=["pdf", 'doc', 'jpg', 'png','txt']

[maxFiles]="3"

(selectedList)="onFileChange($event)"

>

</aava-file-upload>

<p>Single file with left-side positioning</p>

<aava-file-upload

singleFileSelectionPosition="left"

[allowedFormats]=["pdf", 'doc', 'jpg', 'png']"

(selectedList)="onFileChange($event)"

>

</aava-file-upload>

</div>

<div>

<h1>Icon Layout</h1>

<aava-file-upload

layout="icon"

[allowedFormats]=["pdf", 'doc', 'jpg', 'png']"

[maxFiles]="3"

(selectedList)="onFileChange($event)"

>

</aava-file-upload>

</div>

<div>
```

```
<h1>List Layout</h1>

<aava-file-upload
    layout="list"
    [allowedFormats]=["pdf", 'doc', 'jpg', 'png']"
    [maxFiles]="3"
    (selectedList)="onFileChange($event)"

>
</aava-file-upload>
</div>

<div>
<h1>Tags Layout</h1>

<aava-file-upload
    layout="tags"
    [allowedFormats]=["pdf", 'doc', 'jpg', 'png']"
    [maxFiles]="3"
    (selectedList)="onFileChange($event)"

>
</aava-file-upload>
</div>

<div>
<h1>Table Layout</h1>

<aava-file-upload
    (selectedList)="onFilesChanged($event, 'advanced')"
    (deletedList)="onDeletedFiles($event)"
    [allowedFormats]="allowedFormats"
    layout="icon"
    [maxFileSize]="maxFileSize"
    uploaderId="advanced"
    deleteIconPosition="right"
```

```
deleteIconName="trash"
deleteButtonSize="sm"
previewLayout="table"
[previewData]="previewData"
>
</aava-file-upload>
</div>
---
previewData = [
{
fileName: '1.png',
fileSize: '5kb',
fileType: 'png',
},
];
public maxFileSize = 3 * 1024 * 1024; //3MB
public allowedFormats: string[] = [
'jpeg',
'jpg',
'png',
'svg',
'doc',
'docx',
'xlsx',
'txt',
'pdf',
];
onFileChange(files: File[]): void {
console.log('Files changed:', files);
```

```

}

uploadedFiles: Record<string, File[]> = {};
upDisabled = false;

onFilesChanged(files: File[], uploaderId: string) {
  this.uploadedFiles[uploaderId] = [...files];
  this.upDisabled = true;
  console.log(
    `Files list changed for ${uploaderId}:`,
    files.map((f) => f.name)
  );
}

onDeletedFiles(files: UploadFile[]) {
  console.log('Deleted files:', files);
}

```

The component supports multiple layout options to fit different UI contexts and use cases:

default – Standard button and file display layout.

- list – Displays files in a stacked list with remove options.
- tags – Shows files as removable tags, suitable for compact interfaces.
- icon – Drag-and-drop area with rich visual preview and side-by-side file listing.
- Each layout offers unique ways to visualize uploaded files and actions.

Size Variants

```

<div>
  <h1>Extra Small (xs)</h1>
  <aava-file-upload
    componentTitle="XS Upload"
    size="xs"
    uploadButtonSize="xs"
    deleteButtonSize="xs"
    [allowedFormats]=["pdf", 'doc', 'jpg']"
  </aava-file-upload>
</div>

```

```
[maxFiles]="2"
(selectedList)="onFileChange($event)"
>
</aava-file-upload>
</div>
<div>
<h1>Small (sm)</h1>
<aava-file-upload
componentTitle="Small Upload"
size="sm"
uploadButtonSize="sm"
deleteButtonSize="sm"
[allowedFormats]=["pdf", 'doc', 'jpg']"
[maxFiles]="2"
(selectedList)="onFileChange($event)"
>
</aava-file-upload>
</div>
<div>
<h1>Medium (md)</h1>
<aava-file-upload
componentTitle="Medium Upload"
size="md"
uploadButtonSize="md"
deleteButtonSize="md"
[allowedFormats]=["pdf", 'doc', 'jpg']"
[maxFiles]="2"
(selectedList)="onFileChange($event)"
```

```
>
</aava-file-upload>
</div>
<div>
<h1>Large (lg)</h1>
<aava-file-upload
    componentTitle="Large Upload"
    size="lg"
    uploadButtonSize="lg"
    deleteButtonSize="lg"
    [allowedFormats]=["pdf", 'doc', 'jpg']"
    [maxFiles]="2"
    (selectedList)="onFileChange($event)"
>
</aava-file-upload>
</div>
<div>
<h1>Extra Large (xl)</h1>
<aava-file-upload
    componentTitle="XL Upload"
    size="xl"
    uploadButtonSize="xl"
    deleteButtonSize="xl"
    [allowedFormats]=["pdf", 'doc', 'jpg']"
    [maxFiles]="2"
    (selectedList)="onFileChange($event)"
>
</aava-file-upload>
</div>
```

```

<div>
  <h1>Tags Layout with xs button size</h1>
  <aava-file-upload
    layout="tags"
    [allowedFormats]=["pdf", 'doc', 'jpg', 'png"]"
    uploadButtonSize="xs"
    deleteButtonSize="xs"
    [maxFiles]="3"
    (selectedList)="onFileChange($event)"
  >
  </aava-file-upload>
</div>
---
onFileChange(files: File[]): void {
  console.log('Files changed:', files);
}

```

Available Sizes

File Upload buttons support multiple size options to align with your design scale:

- xs (Extra Small)– For dense layouts.
- sm (Small)– Compact option.
- md (Medium)– Default size for general usage.
- lg (Large)– For prominent actions or primary uploads.
- xl (Extra Large)– When emphasizing upload as a major user task.

File Validation

```

<div>
  <h1>Format Validation</h1>
  <p>Only PDF and DOC files allowed</p>
  <aava-file-upload

```

```
componentTitle="PDF/DOC Only"
[allowedFormats]=["pdf", 'doc']
[maxFiles]="3"
(selectedList)="onFileChange($event)"

>
</aava-file-upload>
</div>
<div>
<h1>Size Validation</h1>
<p>Maximum 1MB per file</p>
<aava-file-upload
    componentTitle="Max 1MB Files"
    [allowedFormats]=["pdf", 'doc', 'jpg', 'png"]
    [maxFileSize]="1048576"
    [maxFiles]="3"
    (selectedList)="onFileChange($event)"

>
</aava-file-upload>
</div>
<div>
<h1>Max Files Validation</h1>
<p>Maximum 2 files allowed</p>
<aava-file-upload
    componentTitle="Max 2 Files"
    [allowedFormats]=["pdf", 'doc', 'jpg', 'png"]
    layout="tags"
    [maxFiles]="2"
    (selectedList)="onFileChange($event)"

>
```

```

</aava-file-upload>
</div>
<div>
<h1>Duplicate Detection</h1>
<p>Shows error for duplicate files</p>
<aava-file-upload
    componentTitle="No Duplicates"
    layout="tags"
    [allowedFormats]=["pdf", 'doc', 'jpg', 'png"]"
    [showDuplicateError]="true"
    duplicateErrorText="This file already exists!"
    [maxFiles]="5"
    (selectedList)="onFileChange($event)"
    >
</aava-file-upload>
</div>
---

onFileChange(files: File[]): void {
  console.log('Files changed:', files);
}

Built-in validation ensures that uploaded files meet specific criteria:


- Allowed Formats: Restrict file types with the allowedFormats input.
- File Size Limit: Prevent large files using the maxFileSize property.
- Duplicate Check: Detect and warn users about duplicate uploads.


Validation feedback is displayed inline with error messages.

```

Single vs Multiple

[Error extracting code]

Configure whether users can upload a single file or multiple files at once:

- Usesingle FileMode = truefor a single-file input.

- Keep `itfalseto` allow multi-selection.

The component automatically manages file replacement and count validation.

States & Customization

[Error extracting code]

Customize button variants, labels, colors, and disabled states for consistent UI theming.

The component supports:

- Custom border and divider colors
- Dynamic styles via `customStyles`
- Disabled state to prevent interactions
- Configurable button variants for upload and delete actions

Preview Layouts

[Error extracting code]

Display uploaded files in different preview formats:

- `default`– Shows file cards with color-coded type indicators.
- `table`– Tabular view with configurable columns (`fileName`, `fileSize`, `fileType`, etc.)

Each layout provides structured visibility of uploaded files and supports deletion.

Advanced Features

[Error extracting code]

Explore extended functionality for advanced scenarios:

- Custom table columns for preview layouts.
- Dynamic color mapping for file types.
- Custom upload animations.
- Event emitters (`selectedList`, `deletedList`) for parent integration.

These options make the File Upload component flexible for enterprise-grade workflows.

Icon Customization

[Error extracting code]

Adjust icon styles and placement for both upload and delete actions using these inputs:

- uploadIconName / deleteIconName – Change icon symbols.
- uploadIconPosition / deleteIconPosition – Control left, right, or icon-only layouts.
- tagIcon – Set custom tag icons for tag-based uploads.

This allows seamless integration with your application's visual language.

Features

Upload Methods

- Click to Upload: Traditional file selection via file picker
- Drag and Drop: Intuitive drag-and-drop file upload
- Multiple File Support: Upload single or multiple files
- File Replacement: Replace existing files in single file mode

File Validation

- Format Validation: Restrict file types based on extensions
- Size Validation: Enforce maximum file size limits
- Duplicate Prevention: Prevent duplicate file uploads
- Error Handling: Clear error messages for validation failures

Layout Options

- Inline: Compact form integration
- List: Simple list-based display
- Medium Variant: Enhanced with file tags
- Large Variant: Full-featured with preview
- Customizable: Flexible sizing and theming

User Experience

- Visual Feedback: Clear upload states and progress indicators
- File Preview: Preview uploaded files with metadata
- Responsive Design: Adapts to different screen sizes
- Accessibility: Full keyboard navigation and screen reader support

API Reference

Inputs

[TABLE]

Property	Type	Default	Description
theme	'light' 'dark' 'light'	'light'	Theme variant for the component
uploaderId	string	"	Unique identifier for the uploader
enableAnimation	boolean	false	Enable animation effects
allowedFormats	string[]	['jpeg', 'jpg', 'png', 'svg', 'doc', 'docx', 'xlsx', 'txt', 'pdf']	Allowed file extensions
single FileMode	boolean	false	Restrict to single file upload
maxFiles	number	null	Maximum number of files allowed
componentTitle	string	'Upload File Here'	Title displayed in the upload area
supportedFormatLabel	string	'Supported file formats'	Label for supported formats section
maxFileSize	number	3145728 (3MB)	Maximum file size in bytes
showDialogCloseIcon	boolean	true	Show close icon in dialog mode
showUploadButton	boolean	true	Show upload button
layout	'default' 'inline' 'list' 'md-variant' 'lg-variant' 'default'	'default'	Layout variant for the component
previewLayout	'default' 'table' 'default'	'default'	Preview layout type
previewLayoutHeight	number	0	Custom height for preview layout
singleFileSelectionPosition	'right' 'left' 'right'	'right'	Position of single file selection button
uploadButtonSize	'xs' 'sm' 'md' 'lg' 'xl' 'md'	'md'	Upload button size
uploadButtonLabel	string	'Upload'	Upload button label text
uploadButtonVariant	ButtonVariant	'primary'	Upload button style variant
uploadIconPosition	'left' 'right' 'only' 'left'	'left'	Upload button icon position
uploadIconName	string	'upload'	Upload button icon name
deleteButtonSize	'xs' 'sm' 'md' 'lg' 'xl' 'md'	'md'	Delete button size
deleteButtonLabel	string	'Remove All'	Delete button label text
deleteButtonVariant	ButtonVariant	'secondary'	Delete button style variant
deleteIconPosition	'left' 'right' 'only' 'left'	'left'	Delete button icon position
deleteIconName	string	'circle-x'	Delete button icon name
disabled	boolean	false	Disable the file uploader component

previewData | any | undefined | External preview data
customStyles | Record<string, string> | {} | CSS custom properties override
borderColor | string | '#9ca1aa' | Border color for upload area
ellipses | 'start' | 'end' | 'middle' | 'end' | Ellipses style for long file names
dividerColor | string | '#9ca1aa' | Divider line color
tagIcon | string | 'x' | Icon for removable tags
tableColumns | customColumns[] | Predefined column config | Configurable columns for table layout
fileTypeColors | { [key: string]: string } | undefined | Dynamic colors for file types
border | string | 'var(--fileupload-upload-area-default-border)' | Border style for upload area
[/TABLE]

Outputs

[TABLE]

Event | Type | Description

selectedList | EventEmitter<File[]> | Emitted when a list of files is selected
deletedList | EventEmitter<UploadFile[]> | Emitted when a list of files is deleted
[/TABLE]

Properties

[TABLE]

Property | Type | Description

uploadedFiles | File[] | Array of currently uploaded files
fileUploadedSuccess | boolean | Whether file upload was successful
fileFormatError | boolean | Whether there's a file format error
fileSizeError | boolean | Whether there's a file size error
maxFilesError | boolean | Whether maximum files limit is exceeded
isUploadActive | boolean | Whether upload is currently active
viewAll | boolean | Whether to show all files or truncated list
uniqueId | string | Unique identifier for the uploader
[/TABLE]

Methods

[TABLE]

Method	Parameters	Return	Description
openFileSelector()	None	void	Open file selection dialog
resetUpload()	None	void	Reset upload state and clear files
closeUpload()	None	void	Close upload and reset state
removeNewFile()	file: File	void	Remove specific file from list
getFileExtension()	filename: string	string	Get file extension from filename
toggleViewAll()	None	void	Toggle between truncated and full file list
allowAccepted()	None	string	Get accepted file types string for input

[/TABLE]

Computed Properties

[TABLE]

Property	Type	Description
allowedFormatsList	string[]	List of allowed file formats

[/TABLE]

Layout Variants

Default Layout

The standard file upload interface with drag-and-drop support and file list display.

Inline Layout

Compact uploader designed for form integration with minimal space requirements.

Features:

- Single file display
- Upload button with icon
- File name with remove option
- Minimal footprint

List Layout

Simple list-based file display with upload button and file management.

Features:

- List of uploaded files
- File icons and names
- Individual file removal
- Multiple file support

Medium Variant

Enhanced uploader with file tags and grid layout for better file organization.

Features:

- File count header
- Grid-based file display
- File tags with remove functionality
- Bulk remove all option
- Empty state with format information

Large Variant

Full-featured uploader with comprehensive file management and preview capabilities.

Features:

- Drag-and-drop interface
- File preview panel
- Detailed file information
- File type icons
- Comprehensive error handling
- Bulk file management

File Validation

Format Validation

The component validates file types based on the
allowedFormats

input:

```
// Default allowed formats
```

```
allowedFormats  
= [  
  "jpeg"  
  ,  
  "jpg"  
  ,  
  "png"  
  ,  
  "svg"  
  ,  
  "doc"  
  ,  
  "docx"  
  ,  
  "xlsx"  
  ,  
  "txt"  
  ,  
  "pdf"  
  ,  
]  
;  
// Custom format restriction  
allowedFormats  
= [  
  "pdf"  
  ,
```

```
"doc"  
,
```

"docx"

```
]  
;  
// Only document files
```

Size Validation

File size validation is controlled by the

maxFileSize

input:

// 3MB default

maxFileSize

=

3

*

1024

*

1024

;

// 10MB limit

maxFileSize

=

10

*

1024

*

1024

;

// 1MB limit

```
maxFileSize
```

```
=
```

```
1024
```

```
*
```

```
1024
```

```
;
```

Duplicate Prevention

The component automatically prevents duplicate file uploads by checking:

- File name
- File size
- File content (basic comparison)

Error Handling

File Format Error

Displayed when an unsupported file type is selected:

```
<
```

```
div
```

```
class
```

```
=
```

```
"
```

```
error-message
```

```
"
```

```
role
```

```
=
```

```
"
```

```
alert
```

```
"
```

```
>
```

Invalid file type. Allowed formats: JPEG, JPG, PNG, SVG, DOC, DOCX, XLSX, TXT,

PDF.

</

div

>

File Size Error

Displayed when a file exceeds the maximum size limit:

<

div

class

=

"

error-message

"

role

=

"

alert

"

>

File is too large. Maximum size allowed is 3 MB

</

div

>

Maximum Files Error

Displayed when the maximum file limit is exceeded:

<

div

class

=

```
"  
error-message  
"  
  
role  
=  
"  
  
alert  
"  
  
>  
Maximum of 5 files allowed  
</  
div  
>
```

Best Practices

Design Guidelines

- Layout Selection: Choose appropriate layout variant for your use case
- File Limits: Set reasonable file size and count limits
- Format Restrictions: Clearly communicate supported file types
- Visual Feedback: Provide clear upload states and progress indicators
- Error Handling: Display user-friendly error messages

Accessibility

- Keyboard Navigation: Ensure full keyboard support for all interactions
- Screen Reader Support: Provide clear labels and state announcements
- Drag and Drop: Support both mouse and keyboard file selection
- Error Announcements: Properly announce validation errors
- Focus Management: Clear focus indicators and logical tab order

Performance

- File Validation: Validate files on selection for immediate feedback

- Memory Management: Handle large files appropriately
- Batch Processing: Consider batch uploads for multiple files
- Progress Indicators: Show upload progress for better UX
- Error Recovery: Allow users to retry failed uploads

User Experience

- Clear Instructions: Provide clear upload instructions and format requirements
- Visual Feedback: Show upload states and file information
- File Preview: Allow users to review files before upload
- Bulk Operations: Support bulk file removal and management
- Responsive Design: Ensure uploader works on all device sizes

Integration

- Form Integration: Properly integrate with Angular forms
- Event Handling: Use the comprehensive event system for upload management
- State Management: Coordinate upload state with your application
- File Processing: Handle file processing and storage appropriately
- Error Handling: Implement proper error handling and user feedback

Responsive Behavior

Mobile Adaptations

The file upload component automatically adapts to mobile screens:

- Touch Optimization: Optimized touch targets for mobile interaction
- Mobile Layout: Appropriate layout adjustments for small screens
- File Selection: Mobile-friendly file selection methods
- Error Display: Mobile-optimized error message display

Breakpoint Behavior

- Desktop (>768px): Full upload interface with all features
- Mobile ($\leq 768\text{px}$): Compact layout with optimized spacing
- File Display: Responsive file list and preview

- Button Sizing: Appropriate button sizes for different screens

Content Considerations

- File Names: File names adapt to different screen widths
- Error Messages: Error messages maintain readability on small screens
- Upload Area: Upload area adapts to available space
- File List: File list maintains usability across screen sizes

PROMPT-BAR

The Prompt Bar component is a sophisticated input system designed for modern chat interfaces, AI prompt systems, and messaging applications. It combines a multi-line textarea with customizable action icons, device selection dropdowns, and file attachment capabilities to create a comprehensive input experience.

The

```
<aava-prompt-bar>
```

component provides an intuitive interface for users to compose messages, select target devices, attach files, and interact with various input actions through a clean, organized layout.

How to use

```
import  
{  
  AavaPromptBarComponent  
,  
  PromptIcons  
,  
  DropdownOption  
,  
  uploadedImages  
,  
}  
from  
"@aava/play-core"  
;
```

Interactive Examples

Explore different prompt bar configurations and features with interactive demos.

Basic Usage

The Prompt Bar provides a unified interface for user text input, actions, and attachments.

It supports typing, sending messages, and triggering actions via icons.

Use the basic setup to render the default prompt bar with message input and optional send behavior.

```
<aava-prompt-bar placeholder="Type your message here..." [icons]="allIcons">
```

```
</aava-prompt-bar>
```

```
---
```

```
onMessageSent(message: string) {
```

```
  console.log('Message sent:', message);
```

```
}
```

```
allIcons: PromptIcons[] = [
```

```
{
```

```
  name: 'paperclip',
```

```
  slot: 'icon-start',
```

```
  color: 'green',
```

```
  visible: true,
```

```
  click: () => this.onAttachFile()
```

```
,
```

```
{
```

```
  name: 'send',
```

```
  slot: 'icon-end',
```

```
  color: 'var(--color-text-primary)',
```

```
  visible: true,
```

```
  click: () => this.onAddImage()
```

```
}
```

```
];
```

```
onAttachFile() {
```

```
  console.log('Attach file clicked');
```

```
}
```

```
onAddImage() {
```

```
  console.log('Add image clicked');
```

```
}
```

Sizes

```
<aava-prompt-bar  
size="sm"  
placeholder="Small prompt bar (2 rows)"  
[icons]="allIcons"  
>  
</aava-prompt-bar>  
  
<aava-prompt-bar  
size="md"  
placeholder="Medium prompt bar (3 rows)"  
[icons]="allIcons"  
>  
</aava-prompt-bar>  
  
<aava-prompt-bar  
size="lg"  
placeholder="Large prompt bar (4 rows)"  
[icons]="allIcons"  
>  
</aava-prompt-bar>  
  
---  
allIcons: PromptIcons[] = [  
{  
  name: 'paperclip',  
  slot: 'icon-start',  
  color: 'var(--color-text-primary)',  
  visible: true,  
  click: () => this.onAttachFile()  
,  
{
```

```
name: 'send',
slot: 'icon-end',
color: 'var(--color-text-primary)',
visible: true,
click: () => this.onSend()
}
];

```

Available Sizes

The Prompt Bar comes in three size variants — sm, md, and lg — allowing flexible adaptation to different layouts.

The size affects the overall height, icon dimensions, textarea rows, and tag scaling.

Use smaller sizes for compact UIs and larger ones for more prominent input sections.

- sm (Small)— Slightly larger input for standard compact forms
- md (Medium)— Default size for most use cases (balanced proportions)
- lg (Large)— Larger input for prominent forms or accessibility-focused layouts

Icons & Actions

```
<div>
<h1>Start Icons Only</h1>
<aava-prompt-bar
placeholder="Prompt bar with start icons"
[icons]="startIcons"
>
</aava-prompt-bar>
</div>
<div>
<h1>End Icons Only</h1>
<aava-prompt-bar placeholder="Prompt bar with end icons" [icons]="endIcons">
</aava-prompt-bar>
</div>
```

```
<div>

<h1>Both Start and End Icons</h1>

<aava-prompt-bar

placeholder="Prompt bar with both start and end icons"

[icons]="allIcons"

>

</aava-prompt-bar>

</div>

<div>

<h1>With Select Component</h1>

<aava-prompt-bar

placeholder="Prompt bar with select component"

[icons]="allIcons"

[showSelection]="true"

[deviceOptions]="deviceOptions"

[selectWidth]="'200px'"

selectPlaceholder="Choose device"

>

</aava-prompt-bar>

</div>

<div>

<h1>ng-content with Icons</h1>

<aava-prompt-bar

placeholder="Prompt bar with icons and custom content"

[icons]="allIcons"

>

<div class="custom-content-with-icons">

<button class="action-button" (click)="onCustomAction()">

Custom Action

```

```
</button>

<span class="status-text">Ready</span>

</div>

</aava-prompt-bar>

</div>

<div>

<h1>ng-content Only (No Icons)</h1>

<aava-prompt-bar placeholder="Prompt bar with only custom content">

<div class="custom-content-only">

<button class="primary-button" (click)="onSaveDraft()">Save Draft</button>

<button class="secondary-button" (click)="onClear()">Clear</button>

<button class="primary-button" (click)="onSendMessage()">

    Send Message

</button>

</div>

</aava-prompt-bar>

</div>

<div>

<h1>Custom Third Row Content (ng-content)</h1>

<aava-prompt-bar placeholder="Prompt bar with custom third row content">

<div class="custom-third-row">

<div class="custom-left">

<button class="custom-button">Custom Action</button>

</div>

<div class="custom-center">

<span class="custom-text">Custom Content</span>

</div>

<div class="custom-right">

<button class="custom-button">Send</button>
```

```
</div>
</div>
</aava-prompt-bar>
</div>

---
startIcons: PromptIcons[] = [
{
  name: 'paperclip',
  slot: 'icon-start',
  color: 'var(--color-text-primary)',
  visible: true,
  click: () => this.onAttachFile()
},
{
  name: 'image',
  slot: 'icon-start',
  color: 'var(--color-text-primary)',
  visible: true,
  click: () => this.onAddImage()
};
endIcons: PromptIcons[] = [
{
  name: 'mic',
  slot: 'icon-end',
  color: 'var(--color-text-primary)',
  visible: true,
  click: () => this.onVoiceRecord()
},
```

```
{  
  name: 'send',  
  slot: 'icon-end',  
  color: 'var(--color-text-primary)',  
  visible: true,  
  click: () => this.onSend()  
}  
];  
allIcons: PromptIcons[] = [  
  ...this.startIcons,  
  ...this.endIcons  
];  
deviceOptions: PromptBarOption[] = [  
  { label: 'Desktop', value: 'desktop', icon: 'monitor' },  
  { label: 'Tablet', value: 'tablet', icon: 'tablet' },  
  { label: 'Mobile', value: 'mobile', icon: 'smartphone' }  
];  
onMessageSent(message: string) {  
  console.log('Message sent:', message);  
}  
onIconClick(event: any) {  
  console.log('Icon clicked:', event);  
}  
onAttachFile() {  
  console.log('Attach file clicked');  
}  
onAddImage() {  
  console.log('Add image clicked');  
}
```

```
onVoiceRecord() {  
  console.log('Voice record clicked');  
}  
  
onSend() {  
  console.log('Send clicked');  
}  
  
onCustomAction() {  
  console.log('Custom action clicked');  
}  
  
onSaveDraft() {  
  console.log('Save draft clicked');  
}  
  
onClear() {  
  console.log('Clear clicked');  
}  
  
onSendMessage() {  
  console.log('Send message clicked');  
}
```

Prompt Bars can include action icons on both sides of the input field.

Each icon can be interactive — triggering specific actions like attachments, voice input, or emoji selectors.

Icons are customizable with colors, visibility, and click handlers, making it easy to extend the bar's functionality.

File Upload

```
<div>  
<h1>File Upload Integration (Tags as Thumbnails)</h1>  
<aava-prompt-bar  
placeholder="Type your message with attached files..."  
selectPlaceholder="Choose your device">
```

```

[icons]="icons"
[tags]="tags"
[deviceOptions]="deviceOptions"
[showSelection]="true"
[showImage]="false"
(tagRemoved)="onTagRemoved($event)"

>
</aava-prompt-bar>
</div>
<div>
<h1>File Upload with Image Preview (Images as Previews)</h1>
<aava-prompt-bar
placeholder="Upload images to see preview..."
selectPlaceholder="Choose your device"
[icons]="imageIcons"
[tags]="tagsWithImages"
[deviceOptions]="deviceOptions"
[showSelection]="true"
[showImage]="true"
(tagRemoved)="onTagRemovedWithImages($event)"

>
</aava-prompt-bar>
</div>

```

You can attach files or images to the prompt bar as tags or image previews.

When showImage is enabled, supported image formats (jpg, png, webp, etc.) appear as visual thumbnails, while other files display as tags.

This is ideal for chat interfaces or upload workflows.

States & Configuration

```
<div>
```

```
<h1>Default State</h1>

<aava-prompt-bar placeholder="Type your message here..." [icons]="icons">
</aava-prompt-bar>
</div>
<div>
<h1>Disabled State</h1>
<aava-prompt-bar
placeholder="This prompt bar is disabled"
[disabled]="true"
[icons]="icons"
>
</aava-prompt-bar>
</div>
<div>
<h1>Auto-resize Textarea</h1>
<aava-prompt-bar
placeholder="This textarea will auto-resize as you type..."
[textAreaAutoSize]="true"
[textAreaMaxHeight]="200"
[icons]="icons"
>
</aava-prompt-bar>
</div>
---
icons: PromptIcons[] = [
{
  name: 'send',
  slot: 'icon-end',
  color: 'var(--color-text-primary)',
```

```
visible: true,  
click: () => this.onSend()  
}  
];  
onSend() {  
  console.log('Send clicked');  
}
```

The Prompt Bar supports various states such as disabled, and offers configurable options for width, height, placeholder text, and icon layout.

You can also control auto-resizing, selection dropdowns, and behavior for multi-line input.

These settings help integrate the component seamlessly into different UI contexts.

Prompt Bar Features

- Multi-line Input: Configurable textarea with adjustable rows
- Customizable Icons: Action icons positioned at start and end slots
- Device Selection: Dropdown for selecting target devices or platforms
- File Attachments: Visual badges for uploaded images with removal
- Size Variants: Four size options (xs, sm, md, lg) for different layouts
- Keyboard Shortcuts: Enter to send, Shift+Enter for new lines
- Responsive Design: Mobile-optimized layout with touch-friendly controls
- Auto-scroll: Automatic scrolling to bottom for new content

Features

Multi-line Text Input

The core of the prompt bar is a flexible textarea:

- Configurable Rows: Set the number of visible rows (default: 3)
- Auto-resize: Textarea adjusts to content while maintaining layout
- Placeholder Text: Customizable placeholder for user guidance
- Input Validation: Built-in validation and error handling
- Accessibility: Full keyboard navigation and screen reader support

Icon System

Flexible icon positioning and interaction:

- Slot-based Layout: Icons can be placed in icon-start or icon-end slots
- Customizable Appearance: Size, color, and visibility controls
- Click Handlers: Custom click functions for each icon
- Conditional Display: Show/hide icons based on context
- Responsive Sizing: Automatic size adjustments based on component size

Device Selection

Integrated dropdown for target device selection:

- Custom Options: Define device types with labels and icons
- Visual Indicators: Icons for each device type
- Responsive Layout: Adapts to available space
- Integration Ready: Seamlessly integrates with existing selection systems
- Accessibility: Full keyboard and screen reader support

File Management

Visual file attachment system:

- Image Badges: Display uploaded images as removable tags
- File Information: Show file size, type, and preview
- Removal Controls: Easy file removal with click interaction
- Responsive Layout: Adapts to different screen sizes
- Visual Feedback: Clear indication of attached files

Size Variants

Four predefined size configurations:

- xs (Extra Small): Compact layout for tight spaces
- sm (Small): Standard small layout with balanced proportions
- md (Medium): Default size with optimal spacing
- lg (Large): Spacious layout for prominent displays

API Reference

Input Properties

[TABLE]

Property | Type | Default | Description

messages | PromptMessage[] | [] | Array of previous messages for context

placeholder | string | 'Type a message' | Placeholder text for the input field

disabled | boolean | false | Disable the prompt bar input

icons | PromptIcons[] | [] | Array of icons to display

rows | number | 3 | Number of visible rows in the textarea

deviceOptions | DropdownOption[] | [] | Options for device selection dropdown

showSelection | boolean | false | Show the device selection dropdown

fileOption | string | '' | File upload option configuration

size | 'lg' | 'md' | 'sm' | 'xs' | 'md' | Size variant for the prompt bar

uploadedImages | uploadedImages[] | [] | Array of uploaded image attachments

width | number | 0 | Custom width for the prompt bar

height | number | 0 | Custom height for the prompt bar

textAreaMaxHeight | number | 0 | Maximum height for the textarea

textAreaAutoResize | boolean | false | Auto-resize textarea based on content

[/TABLE]

Output Events

[TABLE]

Event | Type | Description

messageSent | EventEmitter<string> | Emitted when a message is sent

iconClicked | EventEmitter<{ icon: PromptIcons; currentMessage: string }> | Emitted when an icon is clicked

[/TABLE]

Interfaces

interface

PromptIcons

{

```
name
:
string
;
// Icon name/identifier

click
?
:
(
)
=>
void
;

// Optional click handler function

size
?
:
string
;
// Icon size (e.g., '20px')

color
?
:
string
;
// Icon color (hex, rgb, etc.)

slot
:
"icon-start"
```

```
|  
"icon-end"  
;  
// Position slot  
visible  
?  
:  
boolean  
;  
// Whether icon is visible  
}  
interface  
DropdownOption  
{  
label  
:  
string  
;  
// Display text for the option  
value  
:  
string  
;  
// Value associated with the option  
icon  
:  
string  
;  
// Icon name for the option
```

```
}

interface uploadedImages

{
    id
    :
    number
    ;
    // Unique identifier for the image
    label
    :
    string
    ;
    // Display label for the image
    image
    :
    string
    ;
    // Image URL or data
    icon
    :
    string
    ;
    // Icon to display with the image
    size
    :
    string
    ;
    // File size information
```

```
}
```

Methods

```
// Trigger message send programmatically
```

```
triggerSend
```

```
(
```

```
)
```

```
:
```

```
void
```

```
// Get current message content
```

```
getCurrentMessage
```

```
(
```

```
)
```

```
:
```

```
string
```

```
// Get icon size based on component size
```

```
get
```

```
getIconSize
```

```
(
```

```
)
```

```
:
```

```
number
```

```
// Get icons filtered by slot position
```

```
getIconsBySlot
```

```
(
```

```
slot
```

```
:
```

```
'icon-start'
```

```
|
```

```
'icon-end'
```

```
)  
:  
PromptIcons  
[  
]  
// Get size class for styling  
getPromptBarSizeClass  
(  
)  
:  
string  
// Handle message sending  
sendMessage  
(  
)  
:  
void  
// Handle keyboard input  
onKeyPress  
(  
event  
:  
KeyboardEvent  
)  
:  
void  
// Handle input changes  
onInput  
(
```

```
event
:
Event
)
:
void
// Handle icon clicks
onIconClick
(
icon
:
PromptIcons
)
:
void
// Handle file badge removal
removeBadge
(
id
:
number
)
:
void
// Handle device selection changes
onSelectionChange
(
data
:
```

```
any
```

```
)
```

```
:
```

```
void
```

CSS Custom Properties

The prompt bar component uses CSS custom properties for theming:

Container Properties

[TABLE]

Property	Default Value	Description
----------	---------------	-------------

--textbox-background	#ffffff	Background color of the prompt bar
----------------------	---------	------------------------------------

--promptbar-border	1px solid #e2e8f0	Border style of the prompt bar
--------------------	-------------------	--------------------------------

--promptbar-border-radius	8px	Border radius of the prompt bar
---------------------------	-----	---------------------------------

--textbox-textarea-container-padding	16px	Padding for the textarea container
--------------------------------------	------	------------------------------------

--textbox-gap	12px	Gap between elements
---------------	------	----------------------

[/TABLE]

Textarea Properties

[TABLE]

Property	Default Value	Description
----------	---------------	-------------

--textbox-textarea-container-padding	16px	Padding for the textarea
--------------------------------------	------	--------------------------

--textbox-gap	12px	Gap between textarea elements
---------------	------	-------------------------------

[/TABLE]

Badge Properties

[TABLE]

Property	Default Value	Description
----------	---------------	-------------

--badge-default-background	#f3f4f6	Background color for badges
----------------------------	---------	-----------------------------

--promptbar-text-color	#6b7280	Text color for badges
------------------------	---------	-----------------------

[/TABLE]

Responsive Breakpoints

[TABLE]

Breakpoint	Description
max-width: 576px	Mobile layout adjustments
max-width: 360px	Small mobile optimizations

[/TABLE]

Best Practices

Icon Configuration

- Logical Grouping: Group related icons in the same slot
- Consistent Sizing: Use consistent icon sizes within the same component
- Meaningful Colors: Use colors that convey meaning (e.g., green for success)
- Accessibility: Ensure icons have proper alt text and descriptions
- Touch Targets: Maintain adequate touch target sizes for mobile

Device Selection

- Clear Labels: Use descriptive labels for device options
- Relevant Icons: Choose icons that clearly represent each device type
- Logical Ordering: Arrange options in logical order (e.g., Desktop, Tablet, Mobile)
- Default Selection: Provide sensible default device selection
- Validation: Validate device selection before processing

File Management

- File Types: Support common image formats (JPG, PNG, GIF)
- Size Limits: Implement reasonable file size restrictions
- Preview Quality: Provide clear image previews in badges
- Removal Confirmation: Consider confirmation for file removal
- Error Handling: Gracefully handle file upload failures

User Experience

- Keyboard Shortcuts: Provide intuitive keyboard navigation
- Visual Feedback: Clear indication of input states and actions
- Responsive Design: Ensure usability across all device sizes

- Loading States: Show loading indicators for async operations
- Error Messages: Provide helpful error messages and recovery options

Performance

- Icon Optimization: Use optimized icon assets
- Lazy Loading: Load non-critical features on demand
- Event Handling: Efficient event listener management
- Memory Management: Proper cleanup of file references
- Change Detection: Use OnPush strategy for optimal performance

AVATAR

A sophisticated and flexible avatar component designed to display user profiles, status indicators, and visual representations with support for images, text labels, badges, and animated gradient borders. Perfect for user interfaces, chat applications, social platforms, and any system requiring user identification and status display.

How to use

```
import {  
  AavaAvatarsComponent  
}  
from "@aava/play-core";
```

Basic Usage

The most basic implementation with default settings and image support.

Avatar Sizes

Seven distinct sizes to suit different design requirements and use cases.

Available Sizes

- **xss** (Ultra Small): Very compact size for extremely dense layouts
- **xs** (Extra Small): Compact size for dense layouts and lists
- **sm** (Small): Small size for compact interfaces
- **md** (Medium): Standard size for most use cases
- **lg** (Large): Large size for prominent content (default)
- **xl** (Extra Large): Extra large size for emphasis and accessibility
- **xxl** (Ultra Large): Maximum size for hero sections and special emphasis

Avatar Shapes

Two shape variants to match different design systems and preferences.

Shape Features

- Pill: Circular shape with smooth rounded corners
- Square: Modern square shape with subtle border radius
- Responsive: Border radius adapts to size for optimal appearance

Badge Integration

Seamless integration with badge components for notifications and status indicators.

Badge Features

- Status Badges: Visual indicators for online/offline status
- Count Badges: Numeric indicators for notifications
- Custom States: Support for various badge states and colors
- Positioning: Automatic positioning at top-right corner
- Responsive: Badge size adapts to avatar size

Text Labels

Support for both status and profile text labels with flexible positioning.

Text Label Features

- Status Text: Small text for status indicators
- Profile Text: Larger text for user names or titles
- Dual Support: Can display both status and profile text simultaneously
- Typography: Different font sizes for hierarchy
- Alignment: Proper alignment with avatar element

Avatar Initials

Avatar can also support text-based initials

Initials Features

- Custom Initials: Allow developers to provide initials directly
- Background Colors: Configurable solid backgrounds
- Text Styling: Adjustable font size, weight, and color for initials visibility.
- Size Variants: Support for sm, md, lg (and more) to fit different UI contexts.

Accessibility Features

- Screen Reader Support: Proper ARIA labels and descriptions
- Focus Indicators: Clear visual focus indicators
- High Contrast: Enhanced visibility in high contrast mode
- Reduced Motion: Respects user motion preferences
- Alt Text: Support for image alternative text

API Reference

Inputs

[TABLE]

Property | Type | Description

size | 'xxs' | 'xs' | 'sm' | 'md' | 'lg' | 'xl' | 'xxl' | Size of the avatar component

shape | 'pill' | 'square' | Shape variant of the avatar

imageUrl | string | URL of the avatar image

statusText | string | Small text for status indicators

profileText | string | Larger text for user names or titles

badgeState | BadgeState | State of the badge component

badgeSize | BadgeSize | Size of the badge component

badgeCount | number | Numeric value for count badges

additionalText | string | Additional text label for extra information

initials | string | Initials shown in avatar

initialsBackground | string | Background color for initials

initialsColor | string | Text color for initials

customStyles | Record<string, string> | CSS custom properties override

[/TABLE]

CSS Custom Properties

[TABLE]

Property | Default | Description

--avatar-size-us | Dynamic | Ultra small avatar size

--avatar-size-xs | Dynamic | Extra small avatar size

--avatar-size-sm | Dynamic | Small avatar size
--avatar-size-md | Dynamic | Medium avatar size
--avatar-size-lg | Dynamic | Large avatar size
--avatar-size-xl | Dynamic | Extra large avatar size
--avatar-size-ul | Dynamic | Ultra large avatar size
--avatar-border-radius | Dynamic | Border radius for avatar shapes
--avatar-border-radius-us | Dynamic | Border radius for ultra small size
--avatar-border-radius-xs | Dynamic | Border radius for extra small size
--avatar-border-radius-sm | Dynamic | Border radius for small size
--avatar-border-radius-md | Dynamic | Border radius for medium size
--avatar-border-radius-lg | Dynamic | Border radius for large size
--avatar-border-radius-xl | Dynamic | Border radius for extra large size
--avatar-border-radius-ul | Dynamic | Border radius for ultra large size

[/TABLE]

Accessibility Guidelines

Screen Reader Support

- Provide meaningful alt text for avatar images
- Use descriptive labels for avatar purposes
- Include context about user status and information

Visual Design

- Ensure avatar images meet minimum size requirements
- Provide clear visual distinction between different states
- Support high contrast and reduced motion preferences
- Use consistent sizing and spacing across the interface

Best Practices

Design Guidelines

- Appropriate Sizing: Choose sizes based on context and importance
Ultra Small/Extra Small: Use for extremely dense layouts, data tables, and compact interfaces
Small/Medium: Use for standard layouts, navigation, and user lists
Large: Use for featured content, profile headers, and important user

informationExtra Large/Ultra Large: Use for hero sections, prominent displays, and accessibility emphasis

- Consistent Shapes: Use consistent shapes within the same interface
- Clear Hierarchy: Use size and styling to indicate importance
- Status Clarity: Make status indicators clear and unambiguous
- Image Quality: Use high-quality, properly sized images

Performance

- Image Optimization: Optimize avatar images for web delivery
- Lazy Loading: Implement lazy loading for avatar images
- Caching: Cache frequently used avatar images
- Animation Performance: Use hardware-accelerated animations
- Bundle Optimization: Import only needed avatar features

User Experience

- Loading States: Provide loading indicators for remote images
- Fallback Handling: Implement graceful fallbacks for missing images
- Consistent Behavior: Maintain consistent behavior across avatar types
- Responsive Design: Ensure avatars work well on all screen sizes
- Touch Targets: Ensure adequate touch target sizes for mobile

Implementation Considerations

- Image Management: Implement proper image loading and error handling
- State Management: Properly manage avatar states in your application
- Accessibility: Ensure all avatar features are accessible
- Theming: Use CSS custom properties for consistent theming
- Testing: Test avatars across different devices and browsers

Accessibility Implementation

- Semantic HTML: Use proper HTML structure for avatar elements
- ARIA Attributes: Implement appropriate ARIA labels and roles
- Focus Management: Ensure logical focus order and indicators
- Screen Reader Testing: Test with actual screen readers

- Keyboard Testing: Verify complete keyboard navigation flow

BADGE

A smart notification badge component for displaying counts, status indicators, and priority levels. Features intelligent number formatting (9+, 99+, 999+), multiple size variants including dots, semantic color states, and comprehensive accessibility support for enhanced user communication.

How to use

```
import  
{  
  AavaBadgesComponent  
}  
from  
"@aava/play-core"  
;
```

Basic Usage

Simple badge implementations with counts and basic styling.

```
<aava-badges  
[count]="5"  
[state]="'neutral'"  
[size]="'md'"  
(click)="onBadgeClick($event)"  
(keydown.enter)="onBadgeKeyPress($event)"  
(keydown.space)="onBadgeKeyPress($event)"  
tabindex="0"  
role="button"  
aria-label="Badge with count 5"  
>  
</aava-badges>  
<aava-badges  
[count]="3"  
[state]="'high-priority'"
```

```
[size]="'md'"
(click)="onBadgeClick($event)"
(keydown.enter)="onBadgeKeyPress($event)"
(keydown.space)="onBadgeKeyPress($event)"
tabindex="0"
role="button"
aria-label="Badge with count 3"
>
</aava-badges>
<aava-badges
[count]="12"
[state]="'low-priority'"
[size]="'md'"
(click)="onBadgeClick($event)"
(keydown.enter)="onBadgeKeyPress($event)"
(keydown.space)="onBadgeKeyPress($event)"
tabindex="0"
role="button"
aria-label="Badge with count 12"
>
</aava-badges>
<aava-badges
[count]="8"
[state]="'information'"
[size]="'md'"
(click)="onBadgeClick($event)"
(keydown.enter)="onBadgeKeyPress($event)"
(keydown.space)="onBadgeKeyPress($event)"
tabindex="0"
```

```
role="button"
aria-label="Badge with count 8"
>
</aava-badges>
---
onBadgeClick(event: Event): void {
  console.log('Badge clicked:', event);
}
onBadgeKeyPress(event: Event): void {
  console.log('Badge key pressed:', event);
}
```

Sizes

Four size variants to fit different interface requirements and visual hierarchy.

```
<aava-badges
  [count]="1"
  [state]="'high-priority'"
  [size]="'xs'"
  (click)="onBadgeClick($event)"
  tabindex="0"
  role="button"
  aria-label="Badge with count 1 and xs size"
>
</aava-badges>
<aava-badges
  [count]="5"
  [state]="'medium-priority'"
  [size]="'sm'"
  (click)="onBadgeClick($event)"
  tabindex="0"
```

```
role="button"
aria-label="Badge with count 5 and sm size"
>
</aava-badges>
<aava-badges
[count]="15"
[state]="'low-priority'"
[size]="'md'"
(click)="onBadgeClick($event)"
tabindex="0"
role="button"
aria-label="Badge with count 15 and md size"
>
</aava-badges>
<aava-badges
[count]="99"
[state]="'information'"
[size]="'lg'"
(click)="onBadgeClick($event)"
tabindex="0"
role="button"
aria-label="Badge with count 99 and lg size"
>
</aava-badges>
---
onBadgeClick(event: Event): void {
  console.log('Badge clicked:', event);
}
```

Available Sizes

- xs (Extra Small)- Minimal size for very dense interfaces and subtle indicators
- sm (Small)- Compact for dense interfaces and subtle indicators
- md (Medium)- Standard size for most use cases (default)
- lg (Large)- Prominent for important notifications and better accessibility

Variants

Three variant types to serve different use cases.

```
<div class="badge-row">  
  <aava-badges state="high-priority" size="lg" [count]="9"></aava-badges>  
  <aava-badges state="high-priority" size="md" [count]="9"></aava-badges>  
  <aava-badges state="high-priority" size="sm" [count]="9"></aava-badges>  
  <aava-badges state="high-priority" size="xs" [count]="9"></aava-badges>  
</div>  
  
<div class="badge-row">  
  <aava-badges  
    state="medium-priority"  
    size="lg"  
    iconName="clock"  
    iconColor="white"  
    [iconSize]="14"  
  ></aava-badges>  
  <aava-badges  
    state="medium-priority"  
    size="md"  
    iconName="clock"  
    iconColor="white"  
    [iconSize]="12"  
  ></aava-badges>  
  <aava-badges
```

```
state="medium-priority"
size="sm"
iconName="clock"
iconColor="white"
[iconSize]="10"
></aava-badges>
<aava-badges
state="medium-priority"
size="xs"
iconName="clock"
iconColor="white"
[iconSize]="8"
></aava-badges>
</div>
<div class="badge-row">
<aava-badges state="low-priority" size="lg" variant="dots"></aava-badges>
<aava-badges state="low-priority" size="md" variant="dots"></aava-badges>
<aava-badges state="low-priority" size="sm" variant="dots"></aava-badges>
</div>
</div>
```

Available Variants

- Default
- Icon
- Dots

Accessibility

Built-in accessibility features ensuring inclusive user experience.

Accessibility Features

- Keyboard Navigation: Tab navigation and keyboard activation

- Screen Reader Support: Semantic HTML and ARIA attributes
- Focus Indicators: Clear visual focus states for navigation
- Color Independence: Information conveyed beyond color alone
- High Contrast: Enhanced visibility in high contrast modes
- Descriptive Content: Meaningful count and status information

API Reference

Inputs

[TABLE]

Property | Type | Default | Description

state | 'high-priority' | 'medium-priority' | 'low-priority' | 'neutral' | 'information' | 'online' | 'offline' | 'neutral' | Semantic state determining badge color

size | 'lg' | 'md' | 'sm' | 'xs' | 'md' | Size variant of the badge

variant | 'default' | 'dots' | 'default' | Visual variant of the badge

count | number | undefined | Number to display (smart formatting applied)

iconName | string | undefined | Name of icon to display (from ava-icon)

iconColor | string | 'white' | Custom color for the icon

iconSize | number | undefined | Size of the icon in pixels

customStyles | Record<string, string>

[/TABLE]

Properties

[TABLE]

Property | Type | Description

displayCount | string | Formatted count string (includes 9+, 99+, etc.)

badgeClasses | string | Computed CSS classes for badge styling

hasContent | boolean | Whether badge has count or icon content

isDots | boolean | Whether badge is dots variant

isSingleDigit | boolean | Whether count is a single digit (0-9)

[/TABLE]

Methods

[TABLE]

Method | Parameters | Return Type | Description

onKeyPress() | - | void | Handle keyboard interaction (Enter/Space)

[/TABLE]

CSS Custom Properties

[TABLE]

Property | Description

--badge-font | Font for badge text

--badge-weight | Font weight for badge text

--badge-line-height | Line height for badge text

--badge-border-radius | Border radius for badge

--badge-padding | Padding inside badge

--badge-display | CSS display property for badge

--badge-align-items | Vertical alignment of badge content

--badge-justify-content | Horizontal alignment of badge content

--badge-gap | Gap between badge elements

--badge-default-background | Background color for neutral state

--badge-default-text | Text color for neutral state

--badge-default-border | Border for neutral state

--badge-primary-background | Background color for primary state

--badge-primary-text | Text color for primary state

--badge-primary-border | Border for primary state

--badge-secondary-background | Background color for secondary state

--badge-secondary-text | Text color for secondary state

--badge-secondary-border | Border for secondary state

--badge-success-background | Background color for success state

--badge-success-text | Text color for success state

--badge-success-border | Border for success state

--badge-warning-background | Background color for warning state
--badge-warning-text | Text color for warning state
--badge-warning-border | Border for warning state
--badge-error-background | Background color for error state
--badge-error-text | Text color for error state
--badge-error-border | Border for error state
--badge-info-background | Background color for information state
--badge-info-text | Text color for information state
--badge-info-border | Border for information state
--badge-online-background | Background color for online state
--badge-offline-background | Background color for offline state
--badge-size-xs-min-width | Minimum width for extra small badge
--badge-size-xs-height | Height for extra small badge
--badge-size-sm-font | Font for small badge
--badge-size-sm-padding | Padding for small badge
--badge-size-sm-min-width | Minimum width for small badge
--badge-size-sm-height | Height for small badge
--badge-size-md-font | Font for medium badge
--badge-size-md-padding | Padding for medium badge
--badge-size-md-min-width | Minimum width for medium badge
--badge-size-md-height | Height for medium badge
--badge-size-lg-font | Font for large badge
--badge-size-lg-padding | Padding for large badge
--badge-size-lg-min-width | Minimum width for large badge
--badge-size-lg-height | Height for large badge
--badge-solid-background | Background color for solid variant
--badge-solid-text | Text color for solid variant
--badge-solid-border | Border for solid variant
--badge-outline-background | Background color for outline variant

--badge-outline-text	Text color for outline variant
--badge-outline-border	Border for outline variant
--badge-ghost-background	Background color for ghost variant
--badge-ghost-text	Text color for ghost variant
--badge-ghost-border	Border for ghost variant
--badge-dot-size	Size of dot indicator
--badge-dot-border-radius	Border radius for dot (circle)
--badge-dot-margin-right	Margin right for dot
--badge-icon-size	Default size for icons
--badge-icon-color	Default icon color
--badge-icon-margin-right	Margin right for icons
--badge-counter-background	Background color for counter badges
--badge-counter-text	Text color for counter badges
--badge-counter-font	Font for counter badges
--badge-counter-min-width	Minimum width for counter badges
--badge-counter-height	Height for counter badges
--badge-counter-border-radius	Border radius for counter badges
--badge-counter-padding	Padding for counter badges
--badge-font-family	Font family for badge text

[/TABLE]

Best Practices

Design Guidelines

- Use high-priority for urgent notifications requiring immediate attention
- Choose appropriate sizes based on interface density and importance
- Prefer neutral state for simple count displays
- Use dots variant for minimal status indicators without text
- Use icons for status indicators rather than counts
- Consider badge placement to avoid blocking important content

Accessibility

- Ensure badges convey information beyond color alone
- Provide meaningful text content for screen readers
- Use appropriate state colors that meet contrast requirements
- Test keyboard navigation thoroughly
- Consider announcing dynamic count changes to screen readers

Performance

- Avoid frequent count updates that trigger excessive re-renders
- Use OnPush change detection strategy for optimal performance
- Consider debouncing rapid count changes
- Cache computed properties for better performance

Technical Notes

Automatic Expansion

The badge automatically expands its width when displaying multi-character content (like "99+" or "999+") while maintaining circular dimensions for single characters.

Icon vs Count Priority

When both

iconName

and

count

are provided, the count takes priority and the icon is not displayed. Use separate badges for icon + count combinations.

Dots Variant

The dots variant creates simple circular indicators without text content. It's ideal for status indicators, online/offline states, or any minimal visual cue that doesn't require text.

Keyboard Interaction

Badges with tabindex="0" can receive keyboard focus and respond to Enter and Space key presses for custom interactions.

Component Selector

The component uses the selector
ava-badges
(plural) for consistency with the component library naming convention.

ICON

The

```
<aava-icon>
```

component provides a comprehensive icon system built on top of Lucide icons, offering scalable vector graphics with extensive customization options, interaction support, and full accessibility compliance. It supports dynamic sizing, theming, interactive states, and seamless integration into any Angular application.

How to use

```
import {  
  AavalIconComponent  
}  
from "@aava/play-core"  
;
```

Basic Usage

Simple icon implementation using Lucide icon names with default styling.

```
<!-- Basic Icon Usage - Angular Example -->  
<aava-icon iconName="home"></aava-icon>  
<aava-icon iconName="user"></aava-icon>  
<aava-icon iconName="heart"></aava-icon>  
<aava-icon iconName="star"></aava-icon>  
<aava-icon iconName="settings"></aava-icon>
```

Sizes

Flexible sizing options supporting both numeric pixel values and string-based sizing for responsive design.

```
<!-- Icon Sizes - Angular Example -->  
<aava-icon iconName="star" iconSize="16"></aava-icon>  
<aava-icon iconName="star" iconSize="24"></aava-icon>
```

```
<aava-icon iconName="star" iconSize="32"></aava-icon>
<aava-icon iconName="star" iconSize="48"></aava-icon>
<!-- String-based sizing -->
<aava-icon iconName="heart" iconSize="1rem"></aava-icon>
<aava-icon iconName="heart" iconSize="1.5rem"></aava-icon>
<aava-icon iconName="heart" iconSize="2rem"></aava-icon>
```

Size Options

- Numeric values- Direct pixel sizing (e.g.,16,24,32)
- String values- CSS-compatible units (e.g.,'1rem','2em','100%')
- Responsive sizing- Use CSS custom properties for adaptive scaling

Colors

Comprehensive color customization with theme integration and semantic color support.

```
<!-- Icon Colors - Angular Example -->
<aava-icon iconName="palette" iconColor="#ff6b6b"></aava-icon>
<aava-icon iconName="palette" iconColor="#4ecdc4"></aava-icon>
<aava-icon iconName="palette" iconColor="#45b7d1"></aava-icon>
<aava-icon iconName="palette" iconColor="#96ceb4"></aava-icon>
<aava-icon iconName="palette" iconColor="#ffea7"></aava-icon>
```

Color System

- Custom colors- Direct hex, RGB, HSL, or named color values
- CSS custom properties- Theme-aware color tokens
- Semantic colors- Contextual colors based on component state
- Disabled state- Automatic color adaptation for disabled icons

Accessibility

Built-in accessibility features ensuring inclusive user experience and WCAG compliance.

```
<!-- Accessibility Icons - Angular Example -->
<!-- Interactive icons with ARIA labels -->
<aava-icon
```

```
iconName="heart"
cursor="true"
iconColor="#ff6b6b"
aria-label="Like this post"
>
</aava-icon>
<aava-icon
iconName="bookmark"
cursor="true"
iconColor="#45b7d1"
aria-label="Bookmark this article"
>
</aava-icon>
<aava-icon
iconName="share-2"
cursor="true"
iconColor="#4ecdc4"
aria-label="Share this content"
>
</aava-icon>
<!-- Decorative icons with aria-hidden -->
<aava-icon iconName="star" iconColor="#ffeaa7" aria-hidden="true"> </aava-icon>
<!-- Icons with semantic meaning -->
<aava-icon iconName="check-circle" iconColor="#4ecdc4" aria-label="Success">
</aava-icon>
<aava-icon iconName="alert-triangle" iconColor="#ffa726" aria-label="Warning">
</aava-icon>
<aava-icon iconName="x-circle" iconColor="#ff6b6b" aria-label="Error">
</aava-icon>
```

Accessibility Features

- Semantic HTML- Proper button element for interactive icons
- ARIA compliance- Appropriate ARIA attributes and roles
- Keyboard navigation- Full keyboard support for interactive icons
- Screen reader support- Meaningful descriptions and labels
- Focus management- Clear focus indicators and proper focus order
- High contrast- Support for high contrast accessibility modes

API Reference

Inputs

[TABLE]

Property | Type | Default | Description

iconName | string | '' | Lucide icon name to display

color | string | '' | Custom color override (deprecated, use iconColor)

disabled | boolean | false | Whether the icon is disabled

iconColor | string | '#a1a1a1' | Icon color using hex, RGB, HSL, or CSS custom properties

iconSize | number | string | 24 | Icon size in pixels (number) or CSS units (string)

cursor | boolean | false | Whether to show pointer cursor and enable interactions

hoverColor | string | 'none' | Color applied on hover

[/TABLE]

Outputs

[TABLE]

Event | Type | Description

userClick | EventEmitter<Event> | Emitted when interactive icon is clicked

[/TABLE]

Properties

[TABLE]

Property | Type | Description

computedColor | string | Calculated color value based on state

isInteractive | boolean | Whether icon accepts user interactions

isDisabled | boolean | Current disabled state

[/TABLE]

Methods

[TABLE]

Method | Parameters | Description

handleClick() | event: Event | Process click events for interactive icons

[/TABLE]

CSS Custom Properties

[TABLE]

Property | Description

--button-icon-color-disabled | Color used for disabled icon state

--icon-transition | Transition animation for state changes

--icon-hover-color | Color used on hover for interactive icons

--icon-focus-outline | Focus outline styling for accessibility

--icon-focus-outline-offset | Focus outline offset for clear visibility

[/TABLE]

Host Styling

The component applies the following host styles automatically:

display

:

inline-flex

;

align-items

:

center

;

justify-content

```
:  
center  
;  
  
vertical-align  
:  
middle  
;  
  
height  
:  
[iconSize]px  
;  
  
/* Dynamic based on iconSize input */
```

Lucide Icon Integration

The component leverages Lucide icons, providing access to over 1,000 high-quality SVG icons.

Icon Library

- Comprehensive collection- 1,000+ carefully designed icons
- Consistent design- Uniform stroke width and styling
- Scalable vectors- Crisp appearance at any size
- Regular updates- Actively maintained icon library
- Tree-shakable- Only used icons are bundled

Icon Naming

Icons use kebab-case naming convention from the Lucide library:

```
// Common icon examples  
  
iconName  
=  
"home"  
;  
iconName
```

```
=  
"user-circle"  
;
```

```
iconName  
=
```

```
"arrow-right"  
;
```

```
iconName  
=
```

```
"check-circle"  
;
```

```
iconName  
=
```

```
"alert-triangle"  
;
```

Best Practices

Design Guidelines

- Choose appropriate sizes- Use consistent sizing across similar interface elements
- Maintain visual hierarchy- Larger icons for primary actions, smaller for secondary
- Use semantic colors- Colors should convey meaning and match design system
- Consider context- Icon choice should clearly communicate its purpose
- Consistent spacing- Maintain proper spacing around icons in layouts

Accessibility

- Provide meaningful context- Use proper ARIA labels for screen readers
- Interactive indication- Clearly distinguish interactive vs decorative icons
- Keyboard navigation- Ensure all interactive icons are keyboard accessible
- Focus indicators- Maintain clear focus states for navigation
- Color contrast- Ensure sufficient contrast for all color combinations

Performance

- Icon optimization- Lucide icons are automatically optimized SVGs
- Bundle efficiency- Tree-shaking ensures only used icons are bundled
- Avoid excessive sizing- Use reasonable icon sizes to maintain performance
- Minimize color changes- Frequent color updates can impact performance

Integration

- Theme consistency- Use CSS custom properties for consistent theming
- Component composition- Icons work well within buttons, forms, and navigation
- Event handling- Properly handle click events for interactive use cases
- State management- Consider disabled states in your application logic

CHAT-BUBBLE

aava-chat-bubble

renders conversational messages with support for user/AI alignment, avatars, timestamps, and optional action icons per message (copy, like/dislike, speak, reply).

```
import
{
  AavaChatBubbleComponent
}
from
"@aava/play-core"
;
```

Usage

```
<aava-chat-bubble [messages]="messages"></aava-chat-bubble>
---
messages: Message[] = [
{
  sender: 'user',
  senderName: 'Matthew Anderson',
  time: '05:00 pm',
  text: "Hey there! I'm new here and I'm really interested in the concept of tokenized real estate. Can anyone explain how it works?",
  avatar: 'assets/1.svg',
  icons: [],
},
{
  sender: 'ai',
  senderName: 'Matthew Anderson',
  time: '05:01 pm',
  text: 'Hey Matthew, welcome! Tokenized real estate is a way to represent ownership in real estate properties using blockchain technology. Each property is divided into tokens, and each token'
}
```

```
represents a certain fraction of ownership in that property.',  
avatar: 'assets/1.svg',  
icons: [  
  { name: 'thumbs-up', label: 'Like' },  
  { name: 'thumbs-down', label: 'Dislike' },  
,  
,  
];
```

API

Inputs

[TABLE]

Name	Type	Default	Description
------	------	---------	-------------

messages	Array<{ sender: 'user' 'ai'; senderName: string; time: string; text: string; avatar: string; icons: { name: string; label: string }[]; }>	[]	Message array to render. Sender controls alignment and bubble styling.
----------	---	----	--

customStyles	Record<string, string>	{}	Optional custom styles to apply to the chat bubble component dynamically.
--------------	------------------------	----	---

[/TABLE]

Methods

- copyMessage(text: string): Copies message text to clipboard.
- reactToMessage(reaction: 'like' | 'dislike'): Hook to handle reactions.
- replyToMessage(): Hook to trigger reply flow.
- speakMessage(text: string): Uses Speech Synthesis to read message aloud.

CSS Variables

Chat Bubble uses CSS variables for theming:

[TABLE]

Variable	Description
----------	-------------

--chat-bubble-user-background-color	Background color for user messages.
-------------------------------------	-------------------------------------

--chat-bubble-user-text-color	Text color for user messages.
-------------------------------	-------------------------------

--chat-bubble-ai-background-color | Background color for AI messages.
--chat-bubble-ai-text-color | Text color for AI messages.
--chat-bubble-font-family | Font family used in chat bubbles.
--chat-bubble-text | Primary text color used in chat bubbles.
--chat-bubble-line | Line color used for separators or borders.
--chat-bubble-font-weight | Font weight used in chat bubbles.
--chat-bubble-font-size-md | Medium font size used in chat bubbles.
--chat-bubble-font-size-sm | Small font size used in chat bubbles.
--chat-bubble-font-size-xs | Extra small font size used in chat bubbles.

[/TABLE]

Accessibility

- Keyboard focusable action icons.
- Announce message content to assistive tech using readable DOM text.
- Optional speech synthesis via speakMessage.

Best Practices

- Keep messages concise and scannable.
- Include timestamps and sender names when useful.
- Prefer semantic colors via tokens for light/dark themes.

Related

- Avatar
- Icon
- Prompt Bar

CHIP-TAG

A modern, accessible, and highly customizable tag/chip component for displaying labels, categories, status information, and actionable items. Features multiple color variants, sizes, icons, avatars, and interactive capabilities.

How to use

```
import  
{  
  AavaTagComponent  
}  
from  
"@aava/play-core"  
;
```

Basic Usage

Simple tag implementations with labels and basic styling options.

```
<aava-tag type="tag" color="default" size="lg" label="Default Tag"></aava-tag>  
  
<aava-tag  
  type="tag"  
  color="default"  
  size="lg"  
  label="Outlined Tag"  
  [variant]="outlined"  
></aava-tag>  
  
<aava-tag  
  type="tag"  
  color="default"  
  size="lg"  
  label="Pill Tag"  
  [pill]="true"  
></aava-tag>
```

```
<aava-tag  
type="tag"  
color="default"  
size="lg"  
label="Outlined Pill Tag"  
[variant]="outlined"  
[pill]="true"  
></aava-tag>
```

Status Badge

Semantic status badges for different contexts and meanings.

```
<aava-tag type='badge' label="Success" color="success" [size]="'lg'"></aava-tag>  
<aava-tag type='badge' label="Warning" color="warning" [size]="'lg'"></aava-tag>  
<aava-tag type='badge' label="Error" color="error" [size]="'lg'"></aava-tag>  
<aava-tag type='badge' label="Info" color="info" [size]="'lg'"></aava-tag>  
  
---  
onTagClick(tagName: string): void {  
  console.log(`#${tagName} tag clicked!`);  
}
```

Available Status Badge Colors

- Success: Green for positive states and confirmations
- Warning: Orange for caution and attention states
- Error: Red for error states and critical information
- Info: Blue for informational content
- Custom: User-defined colors via CSS custom properties

Variants

Filled and outlined style variants for different visual emphasis.

[Error extracting code]

Style Variants

- Filled: Solid background with high contrast (default)
- Outlined: Transparent background with colored border and text

Sizes

Three size options to fit different interface requirements and hierarchy.

[Error extracting code]

Available Sizes

- xs (Extra Small)- Extra compact size for very dense interfaces
- sm (Small)- Compact size for dense interfaces
- md (Medium)- Standard size for most use cases (default)
- lg (Large)- Prominent size for primary actions
- xl (Extra Large)- Extra large size for hero sections and CTAs

Icons

Icon integration with positioning and customization options.

[Error extracting code]

Icon Features

- Positioning: Start (leading) or end (trailing) positions
- Custom Colors: Override icon colors independently
- Size Adaptation: Icons automatically scale with tag size
- Icon Library: Integration with ava-icon component system

Avatars

User avatar support for person tags and user identification.

[Error extracting code]

Avatar Types

- Image Avatars: Profile pictures from URLs
- Initial Avatars: Text-based avatars with user initials

Interactive Tags

Clickable and removable tags with event handling and state management.

[Error extracting code]

Interactive Features

- Clickable: Click handlers for tag selection and navigation
- Removable: Close button with removal events
- Hover Effects: Visual feedback for interactive states
- Keyboard Support: Enter key activation for accessibility
- Disabled State: Non-interactive disabled appearance

Custom Tags

Custom color tags with user-defined styling using CSS custom properties.

[Error extracting code]

Custom Color Implementation

Create custom colored tags by setting the

color

prop to

"custom"

and providing custom CSS properties through the

customStyle

input:

- --tag-custom-bg: Background color for the tag
- --tag-custom-color: Text color for the tag
- --tag-custom-border: Border styling for the tag

Accessibility

Built-in accessibility features ensuring inclusive user experience.

Accessibility Features

- Keyboard Navigation: Tab navigation and Enter activation
- ARIA Support: Proper roles and attributes for screen readers
- Focus Management: Clear visual focus indicators
- Disabled States: Proper disabled state communication

- Color Contrast: WCAG-compliant color combinations
- Screen Reader: Descriptive labels and state announcements

API Reference

Inputs

[TABLE]

Property	Type	Default	Description
label	string	"	Text label for the tag
color	'default' 'primary' 'success' 'warning' 'error' 'info' 'custom' 'default'		Color variant for the tag
variant	'filled' 'outlined' 'filled'		Visual style variant
size	'xs' 'sm' 'md' 'lg' 'xl' 'sm'		Size of the tag component
pill	boolean	false	Whether to use pill-shaped (fully rounded) design
removable	boolean	false	Whether to show remove button
disabled	boolean	false	Whether the tag is disabled
icon	string undefined		Icon name for ava-icon component
iconPosition	'start' 'end' 'start'		Position of the icon relative to label
avatar	string undefined		Avatar image URL or initials text
iconColor	string undefined		Custom color for icons
customStyle	Record<string, string> undefined		Custom CSS custom properties
customClass	string undefined		Additional CSS class names
type	'badge' 'tag' 'tag'		Component type: tag or status badge

[/TABLE]

Outputs

[TABLE]

Event	Type	Description
clicked	EventEmitter<void>	Emitted when tag is clicked (if clickable)
removed	EventEmitter<void>	Emitted when remove button is clicked

[/TABLE]

Properties

[TABLE]

Property | Type | Description

clickable | boolean | Read-only property indicating if tag has click handlers

[/TABLE]

Methods

[TABLE]

Method | Parameters | Return Type | Description

onClick() | - | void | Handle tag click events

onRemove(event: Event) | event: Event | void | Handle remove button clicks

[/TABLE]

CSS Custom Properties

[TABLE]

Property | Description

--tags-filled-background | Background color for filled variant

--tags-filled-text | Text color for filled variant

--tags-filled-border | Border color for filled variant

--tags-filled-primary-background | Background color for primary filled variant

--tags-filled-primary-text | Text color for primary filled variant

--tags-filled-primary-border | Border color for primary filled variant

--tags-filled-success-background | Background color for success filled variant

--tags-filled-success-text | Text color for success filled variant

--tags-filled-success-border | Border color for success filled variant

--tags-filled-warning-background | Background color for warning filled variant

--tags-filled-warning-text | Text color for warning filled variant

--tags-filled-warning-border | Border color for warning filled variant

--tags-filled-error-background | Background color for error filled variant

--tags-filled-error-text | Text color for error filled variant

--tags-filled-error-border | Border color for error filled variant

--tags-filled-info-background | Background color for info filled variant
--tags-filled-info-text | Text color for info filled variant
--tags-filled-info-border | Border color for info filled variant
--tags-filled-custom-background | Background color for custom filled variant
--tags-filled-custom-text | Text color for custom filled variant
--tags-filled-custom-border | Border color for custom filled variant
--tags-outlined-background | Background color for outlined variant
--tags-outlined-text | Text color for outlined variant
--tags-outlined-border | Border color for outlined variant
--tags-outlined-primary-text | Text color for primary outlined variant
--tags-outlined-primary-border | Border color for primary outlined variant
--tags-outlined-success-text | Text color for success outlined variant
--tags-outlined-success-border | Border color for success outlined variant
--tags-outlined-warning-text | Text color for warning outlined variant
--tags-outlined-warning-border | Border color for warning outlined variant
--tags-outlined-error-text | Text color for error outlined variant
--tags-outlined-error-border | Border color for error outlined variant
--tags-outlined-info-text | Text color for info outlined variant
--tags-outlined-info-border | Border color for info outlined variant
--tags-outlined-custom-text | Text color for custom outlined variant
--tags-outlined-custom-border | Border color for custom outlined variant
--tags-focus-background | Background color for focus state
--tags-focus-text | Text color for focus state
--tags-focus-border | Border color for focus state
--tags-focus-outline | Outline style for focus state
--tags-disabled-background | Background color for disabled state
--tags-disabled-text | Text color for disabled state
--tags-disabled-border | Border color for disabled state

--tags-disabled-cursor | Cursor style when tag is disabled
--tags-pill-border-radius | Border radius for pill variant
--tags-removable-button-background | Background color for remove button
--tags-removable-button-text | Text color for remove button
--tags-removable-button-size | Size of the remove button icon
--tags-removable-button-border-radius | Border radius for remove button
--tags-removable-button-padding | Padding for remove button
--tags-removable-button-active-background | Background color for active remove button
--tags-removable-button-active-text | Text color for active remove button
--tags-avatar-bg | Background color for avatar
--tags-avatar-initials-bg | Background color for avatar initials
--tags-avatar-initials-color | Text color for avatar initials
--tags-avatar-size | Size of avatar
--tags-avatar-font-size | Font size for avatar initials
--tags-avatar-margin | Margin around avatar
--tag-custom-bg | Custom background color
--tag-custom-color | Custom text color
--tag-custom-border | Custom border color
--tags-xs-font-family | Font family for xs size tags
--tags-xs-font-size | Font size for xs size tags
--tags-xs-font-weight | Font weight for xs size tags
--tags-xs-line-height | Line height for xs size tags
--tags-xs-padding | Padding for xs size tags
--tags-xs-border-radius | Border radius for xs size tags
--tags-sm-font-family | Font family for sm size tags
--tags-sm-font-size | Font size for sm size tags
--tags-sm-font-weight | Font weight for sm size tags
--tags-sm-line-height | Line height for sm size tags
--tags-sm-padding | Padding for sm size tags

--tags-sm-border-radius | Border radius for sm size tags
--tags-md-font-family | Font family for md size tags
--tags-md-font-size | Font size for md size tags
--tags-md-font-weight | Font weight for md size tags
--tags-md-line-height | Line height for md size tags
--tags-md-padding | Padding for md size tags
--tags-md-border-radius | Border radius for md size tags
--tags-lg-font-family | Font family for lg size tags
--tags-lg-font-size | Font size for lg size tags
--tags-lg-font-weight | Font weight for lg size tags
--tags-lg-line-height | Line height for lg size tags
--tags-lg-padding | Padding for lg size tags
--tags-lg-border-radius | Border radius for lg size tags
--tags-xl-font-family | Font family for xl size tags
--tags-xl-font-size | Font size for xl size tags
--tags-xl-font-weight | Font weight for xl size tags
--tags-xl-line-height | Line height for xl size tags
--tags-xl-padding | Padding for xl size tags
--tags-xl-border-radius | Border radius for xl size tags
--tags-icon-gap | Default gap between icon and text
--tags-xl-right-icon-gap | Right icon gap for xl size tags
--tags-lg-right-icon-gap | Right icon gap for lg size tags
--tags-md-right-icon-gap | Right icon gap for md size tags
--tags-sm-right-icon-gap | Right icon gap for sm size tags
--tags-xs-right-icon-gap | Right icon gap for xs size tags
--tags-xl-left-icon-gap | Left icon gap for xl size tags
--tags-lg-left-icon-gap | Left icon gap for lg size tags
--tags-md-left-icon-gap | Left icon gap for md size tags
--tags-sm-left-icon-gap | Left icon gap for sm size tags

--tags-xs-left-icon-gap | Left icon gap for xs size tags
--tags-avatar-color | Text color for avatar
--tags-remove-button-disabled-color | Color for disabled remove button
[/TABLE]

Best Practices

Design Guidelines

- Use consistent colors for similar concepts across your application
- Choose appropriate sizes based on interface hierarchy and density
- Prefer outlined variants for secondary information
- Use removable tags for user-manageable content
- Consider pill shape for status indicators and badges

Accessibility

- Always provide meaningful labels that describe the tag purpose
- Use color variants semantically (success for positive states, etc.)
- Ensure sufficient color contrast for all variants
- Provide keyboard navigation for interactive tags
- Use ARIA labels for complex tag interactions

Performance

- Avoid frequent color or style changes that trigger repaints
- Use CSS custom properties for dynamic theming
- Consider virtualization for large tag lists
- Batch tag operations for better performance

TEXT

A component for displaying text, either styled or unstyled.

TABLE

The
DataGridComponent

provides a comprehensive data table solution with advanced features including sorting, filtering, custom cell templates, and responsive design. It uses a flexible column definition system with content projection for maximum customization.

How to use

Import the component and its directives, then define your table structure with custom templates.

```
import  
{  
    AavaDataGridComponent  
,  
    AvaColumnDefDirective  
,  
    AvaHeaderCellDefDirective  
,  
    AvaCellDefDirective  
,  
    AavaTagComponent  
,  
}  
from  
"@aava/play-core"  
;
```

Basic Usage

Simple table with basic data display and column definitions.

```
<div class="demo-page">  
    <!-- Demo Content -->  
    <div class="demo-content">
```

```
<div class="container">
  <!-- Employee Table Section -->
  <div class="demo-section">
    <div class="table-container">
      <aava-data-grid
        [dataSource]="basicData"
        [displayedColumns]="displayedColumns"
        class="styled-data-grid">
        >
        <ng-container avaColumnDef="name">
          <ng-container *avaHeaderCellDef>
            <div class="header-cell">
              <span class="header-text">Employee Name</span>
            </div>
          </ng-container>
          <ng-container *avaCellDef="let row">
            <div class="data-cell name-cell">
              <span class="employee-name">{{ row.name }}</span>
            </div>
          </ng-container>
        </ng-container>
        <ng-container avaColumnDef="email">
          <ng-container *avaHeaderCellDef>
            <div class="header-cell">
              <span class="header-text">Email Address</span>
            </div>
          </ng-container>
          <ng-container *avaCellDef="let row">
            <div class="data-cell email-cell">
```

```
<span class="email-text">{{ row.email }}</span>
</div>
</ng-container>
</ng-container>
<ng-container avaColumnDef="department">
<ng-container *avaHeaderCellDef>
<div class="header-cell">
<span class="header-text">Department</span>
</div>
</ng-container>
<ng-container *avaCellDef="let row">
<div class="data-cell department-cell">
<span class="department-badge">{{ row.department }}</span>
</div>
</ng-container>
</ng-container>
<ng-container avaColumnDef="status">
<ng-container *avaHeaderCellDef>
<div class="header-cell">
<span class="header-text">Status</span>
</div>
</ng-container>
<ng-container *avaCellDef="let row">
<div class="data-cell status-cell">
<aava-tag
  [label]="row.status"
  [color]="getStatusColor(row.status)"
  size="sm"
></aava-tag>
```

```
</div>
</ng-container>
</ng-container>
</aava-data-grid>
</div>
</div>
</div>
</div>
</div>
</div>
---
basicData = [
{
id: 1,
name: 'Alice Johnson',
email: 'alice.johnson@example.com',
department: 'Engineering',
status: 'Active',
},
{
id: 2,
name: 'Bob Smith',
email: 'bob.smith@example.com',
department: 'Marketing',
status: 'Active',
},
{
id: 3,
name: 'Carlos Martinez',
email: 'carlos.martinez@example.com',
```

```
        department: 'Sales',
        status: 'Pending',
    },
{
    id: 4,
    name: 'Diana Lee',
    email: 'diana.lee@example.com',
    department: 'Engineering',
    status: 'Inactive',
},
{
    id: 5,
    name: 'Ethan Brown',
    email: 'ethan.brown@example.com',
    department: 'HR',
    status: 'Active',
},
];
displayedColumns = ['name', 'email', 'department', 'status'];
/***
 * Get the appropriate color for status tags
 */
getStatusColor(
    status: string
): 'success' | 'warning' | 'error' | 'info' | 'default' {
    switch (status.toLowerCase()) {
        case 'active':
            return 'success';
        case 'pending':
```

```

return 'warning';

case 'inactive':
    return 'error';

default:
    return 'default';
}

}

```

Sorting

Table with sortable columns and visual sort indicators.

```

<div class="demo-content">

<div class="demo-section">

<div class="demo-card">

<div class="card-content">

<aava-data-grid

[dataSource]="employeeData"

[displayedColumns]="displayedColumns"

>

<ng-container avaColumnDef="name" [sortable]="true">

<ng-container *avaHeaderCellDef>Employee Name</ng-container>

<ng-container *avaCellDef="let row">{{ row.name }}</ng-container>

</ng-container>

<ng-container avaColumnDef="position" [sortable]="true">

<ng-container *avaHeaderCellDef>Position</ng-container>

<ng-container *avaCellDef="let row"

>{{ row.position }}</ng-container>

</ng-container>

<ng-container avaColumnDef="salary" [sortable]="true">

<ng-container *avaHeaderCellDef>Annual Salary</ng-container>

```

```
<ng-container *avaCellDef="let row"
>${{ row.salary | number }}</ng-container>
>
</ng-container>
<ng-container avaColumnDef="experience" [sortable]="true">
<ng-container *avaHeaderCellDef>Experience (Years)</ng-container>
<ng-container *avaCellDef="let row"
>{{ row.experience }} years</ng-container>
>
</ng-container>
<ng-container avaColumnDef="joinDate" [sortable]="true">
<ng-container *avaHeaderCellDef>Join Date</ng-container>
<ng-container *avaCellDef="let row"
>{{ row.joinDate | date }}</ng-container>
>
</ng-container>
<ng-container avaColumnDef="department">
<ng-container *avaHeaderCellDef>Department</ng-container>
<ng-container *avaCellDef="let row"
>{{ row.department }}</ng-container>
>
</ng-container>
</aava-data-grid>
</div>
</div>
</div>
</div>
---
```

```
employeeData = [
{
    id: 1,
    name: "Alice Johnson",
    position: "Senior Developer",
    salary: 95000,
    joinDate: "2020-03-15",
    experience: 8,
    department: "Engineering",
},
{
    id: 2,
    name: "Bob Smith",
    position: "Marketing Manager",
    salary: 75000,
    joinDate: "2019-07-22",
    experience: 6,
    department: "Marketing",
},
{
    id: 3,
    name: "Carlos Martinez",
    position: "Sales Representative",
    salary: 55000,
    joinDate: "2021-11-08",
    experience: 3,
    department: "Sales",
},
{
```

```
id: 4,  
name: "Diana Lee",  
position: "UX Designer",  
salary: 70000,  
joinDate: "2020-09-12",  
experience: 5,  
department: "Design",  
},  
{  
id: 5,  
name: "Ethan Brown",  
position: "Data Analyst",  
salary: 65000,  
joinDate: "2022-01-30",  
experience: 2,  
department: "Analytics",  
},  
{  
id: 6,  
name: "Fiona Green",  
position: "Project Manager",  
salary: 85000,  
joinDate: "2018-05-10",  
experience: 9,  
department: "Operations",  
},  
{  
id: 7,  
name: "George Wang",
```

```
position: "DevOps Engineer",
salary: 90000,
joinDate: "2019-12-03",
experience: 7,
department: "Engineering",
},
{
id: 8,
name: "Hannah Kim",
position: "Content Writer",
salary: 45000,
joinDate: "2021-08-15",
experience: 1,
department: "Marketing",
},
];
displayedColumns = ["name", "position", "salary", "experience", "joinDate"];
salesData = [
{ month: "January", revenue: 125000, orders: 340, conversion: 3.2 },
{ month: "February", revenue: 135000, orders: 385, conversion: 3.8 },
{ month: "March", revenue: 142000, orders: 420, conversion: 4.1 },
{ month: "April", revenue: 128000, orders: 365, conversion: 3.5 },
{ month: "May", revenue: 155000, orders: 445, conversion: 4.3 },
{ month: "June", revenue: 168000, orders: 478, conversion: 4.6 },
];
salesColumns = ["month", "revenue", "orders", "conversion"];
```

Filtering

Advanced filtering capabilities with multiple filter conditions and operators.

[Error extracting code]

Features

Flexible Column System

- Content projection-based column definitions
- Custom header and cell templates
- Configurable sorting and filtering per column
- Dynamic column visibility

Advanced Sorting

- Multi-column sorting support
- Visual sort indicators (ascending/descending)
- Configurable sort behavior per column
- Sort state management

Powerful Filtering

- Multiple filter conditions and operators
- Real-time filtering with search
- Filter panel with advanced options
- Clear and apply filter actions

Custom Templates

- Flexible cell content templates
- Custom header templates
- Template context with row data and index
- Support for complex cell content

Responsive Design

- Horizontal scrolling for wide tables
- Mobile-friendly design
- Adaptive column sizing
- Touch-optimized interactions

Performance Optimized

- OnPush change detection strategy
- Efficient data handling
- Optimized rendering
- Memory management

API Reference

Inputs

[TABLE]

Property | Type | Default | Description

dataSource | any[] | [] | Array of data objects to display in the table

displayedColumns | string[] | [] | Array of column names to display

[/TABLE]

Outputs

[TABLE]

Property | Type | Description

dataSorted | EventEmitter<any[]> | Emitted when data is sorted with sorted data

[/TABLE]

Directives

[TABLE]

Property | Type | Default | Description

avaColumnDef | string | - | Column name/identifier (required)

sortable | boolean | false | Enable sorting for this column

filter | boolean | false | Enable filtering for this column

[/TABLE]

[TABLE]

Property | Type | Description

Template | TemplateRef<any> | Template for custom header cell content

[/TABLE]

[TABLE]

Property | Type | Description

Template | TemplateRef<any> | Template for custom cell content with context

[/TABLE]

Interfaces

interface

 FilterCondition

 {

 label

 :

 string

 ;

 // Display label for filter condition

 value

 :

 string

 ;

 // Value for filter condition

}

Methods

[TABLE]

Method | Parameters | Description

onSort() | column: AvaColumnDefDirective | Handle column sorting

applySort() | None | Apply current sort to data

applyFilter() | columnName: string, event: Event | Apply filter to specific column

clearFilter() | columnName: string, event: any | Clear filter for specific column

openPanel() | columnName: string, event: any | Open filter panel for column

checkForOpen() | columnName: string | Check if filter panel is open for column

[/TABLE]

Properties

[TABLE]

Property | Type | Description

sortColumn | string | null | Currently sorted column

sortDirection | 'asc' | 'desc' | '' | Current sort direction

sortedData | any[] | Currently sorted and filtered data

filterColumn | Array<{column: string, type: string, value: any, open: boolean}> | Active filters

defaultFilterConditions | FilterCondition[] | Available filter conditions

[/TABLE]

CSS Custom Properties

The component uses CSS custom properties for dynamic styling:

Container Properties

[TABLE]

Property | Description

--grid-font-family-body | Font family for table content

--grid-text-color | Text color for table content

--grid-background-color-odd | Background color for odd rows

--grid-background-color-even | Background color for even rows

--grid-border | Border color for grid elements

[/TABLE]

Table Properties

[TABLE]

Property | Description

--table-border | Border color for table elements

[/TABLE]

CSS Classes

The component uses CSS classes for styling and state management:

Container Classes

[TABLE]

Class | Description

.ava-data-table-wrapper | Main table container

.data-table-wrapper | Inner table wrapper with scrolling

.ava-data-table | Main table element

[/TABLE]

Cell Classes

[TABLE]

Class | Description

.cell-wrapper | Header cell content wrapper

.grid-column-container | Column header container

.filter | Filter icon container

.filter-wrapper | Filter panel container

.default-filter-actions | Filter action buttons container

.cell-link | Link styling within cells

[/TABLE]

State Classes

[TABLE]

Class | Description

.sort-icon | Sort indicator icon

Various pseudo-classes | Hover and focus states

[/TABLE]

Best Practices

Data Structure

- Use consistent data structure across all rows
- Ensure column names match displayedColumns array
- Provide meaningful default values for missing data
- Optimize data for sorting and filtering

Column Definitions

- Use descriptive column names
- Enable sorting only for relevant columns
- Enable filtering for searchable data
- Provide meaningful header labels

Custom Templates

- Keep cell templates simple and focused
- Use template context for row data access
- Implement proper error handling in templates
- Consider accessibility in custom content

Performance

- Limit data size for optimal performance
- Use OnPush change detection strategy
- Implement virtual scrolling for large datasets
- Optimize filter and sort operations

Accessibility

- Provide proper ARIA labels
- Ensure keyboard navigation support
- Use semantic HTML structure
- Maintain color contrast ratios

Responsive Design

- Test table on various screen sizes
- Implement horizontal scrolling for wide tables
- Consider mobile-specific interactions
- Optimize touch targets for mobile

Accessibility

ARIA Support

- Proper table semantics

- Sort and filter announcements
- Screen reader friendly navigation
- Status updates for dynamic content

Keyboard Navigation

- Tab navigation through table elements
- Arrow key navigation between cells
- Enter/Space activation for actions
- Escape key for closing panels

Focus Management

- Clear focus indicators
- Logical tab order
- Focus restoration after actions
- Focus trapping in modals/panels

Screen Reader Support

- Descriptive labels for actions
- Context information for data
- Status announcements
- Clear navigation structure

Browser Support

- Modern Browsers: Full support for all features
- CSS Grid/Flexbox: Required for layout
- ES6+ Features: Required for component functionality
- Template Ref: Required for content projection
- Change Detection: OnPush strategy support

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"  
;
```

Note

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

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>
---
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>
</div>
```

```
</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(), and selectItems() methods
- Event Handling: Comprehensive selection change events with detailed information

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
}

```

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

API Reference

Inputs

[TABLE]

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

[/TABLE]

ListItemsComponent Inputs

[TABLE]

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)

[/TABLE]

Outputs

[TABLE]

Event | Type | Description

onOptionSelected | EventEmitter<ListItem> | Emitted when an item is selected

onSelectionChanged | EventEmitter<ListSelectionEvent> | Emitted when selection changes

onButtonClick | EventEmitter<ListButtonClickEvent> | Emitted when an item button is clicked

onIconClick | EventEmitter<{item: ListItem, event: Event}> | Emitted when an item icon is clicked

[/TABLE]

ListItemsComponent Outputs

[TABLE]

Event | Type | Description

itemClick | EventEmitter<void> | Emitted when the item is clicked

[/TABLE]

Properties

[TABLE]

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

[/TABLE]

Methods

[TABLE]

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
[/TABLE]

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  
?  
:  
ListButtonItem  
[  
]  
;  
disabled  
?  
:  
boolean  
;  
data  
?  
:  
any  
;  
}
```

ListButtonItem

```
interface  
ListButtonItem  
{  
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
:

ListIItem
;

}
```

ListButtonClickEvent

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

ListItemSize

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

[TABLE]

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

[/TABLE]

[TABLE]

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

[/TABLE]

[TABLE]

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

[/TABLE]

[TABLE]

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

[/TABLE]

[TABLE]

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

[/TABLE]

Token Override Example

/* Custom list theming */

.my-compact-list

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

```
px
;
--list-buttons-gap
:
4
px
;
}
```

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 the `aava-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

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

FILTER

A comprehensive filter component that provides both single and multi-selection modes, grouped filter options, and flexible positioning. Built with accessibility in mind and designed for complex data filtering scenarios in data tables, search interfaces, and content management systems.

How to use

```
import {  
  AavaFilterComponent  
}  
from "@aava/play-core"  
;
```

Note

: The Filter component is standalone and includes all necessary dependencies for checkbox, icon, and common modules.

Basic Usage

Simple filter implementation with multi-selection support and grouped options.

```
<aava-filter  
  size="md"  
  title="Filter Products"  
  [filterGroups]="filterGroups"  
  [showBadge]="true"  
  (filterChange)="onFilterChange($event)"  
  (clearAll)="onClearAllFilters()">  
</aava-filter>  
  
<div class="product-card" *ngFor="let product of filteredProducts">  
  <h4>{{ product.name }}</h4>  
  <p>Category: {{ product.category }}</p>  
  <p class="price">${{ product.price }}</p>
```

```
</div>

---

filterGroups = [
{
  id: 'category',
  title: 'Category',
  options: [
    { id: 'electronics', label: 'Electronics', value: 'electronics' },
    { id: 'clothing', label: 'Clothing', value: 'clothing' },
    { id: 'books', label: 'Books', value: 'books' },
    { id: 'home', label: 'Home & Garden', value: 'home' },
  ],
},
{
  id: 'price',
  title: 'Price Range',
  options: [
    { id: 'under-50', label: 'Under $50', value: { min: 0, max: 50 } },
    { id: '50-100', label: '$50 - $100', value: { min: 50, max: 100 } },
    {
      id: '100-200',
      label: '$100 - $200',
      value: { min: 100, max: 200 },
    },
  ],
},
];
products = [
{ id: 1, name: 'Laptop', category: 'electronics', price: 999 },
```

```
{ id: 2, name: 'Smartphone', category: 'electronics', price: 699 },  
{ id: 3, name: 'T-Shirt', category: 'clothing', price: 25 },  
{ id: 4, name: 'Jeans', category: 'clothing', price: 45 },  
{ id: 5, name: 'Novel', category: 'books', price: 15 },  
{ id: 6, name: 'Cookbook', category: 'books', price: 35 },  
{ id: 7, name: 'Coffee Table', category: 'home', price: 299 },  
{ id: 8, name: 'Garden Chair', category: 'home', price: 89 },  
];  
  
filteredProducts = [...this.products];  
  
activeFilters: Record<string, FilterOption[]> = {};  
  
onFilterChange(filters: Record<string, FilterOption[]>): void {  
  console.log('Filter changed:', filters);  
  this.activeFilters = filters;  
  this.applyFilters();  
}  
  
onClearAllFilters(): void {  
  console.log('All filters cleared');  
  this.activeFilters = {};  
  this.filteredProducts = [...this.products];  
}  
  
private applyFilters(): void {  
  this.filteredProducts = this.products.filter((product) => {  
    // Category filter  
    if (  
      this.activeFilters['category'] &&  
      this.activeFilters['category'].length > 0  
    ) {  
      const categoryMatch = this.activeFilters['category'].some(  
        (f: FilterOption) => f.value === product.category  
      );  
      if (categoryMatch) return true;  
    }  
    return false;  
  });  
}
```

```
);

if (!categoryMatch) return false;

}

// Price filter

if (

this.activeFilters['price'] &&
this.activeFilters['price'].length > 0
){

const priceMatch = this.activeFilters['price'].some(
(f: FilterOption) => {

const priceRange = f.value as PriceRange;

return (
product.price >= priceRange.min && product.price <= priceRange.max
);

}
);

if (!priceMatch) return false;

}

return true;
});

}

getActiveFiltersCount(): number {
let count = 0;

Object.values(this.activeFilters).forEach((group) => {

if (Array.isArray(group)) {

count += group.length;

}

});
}

return count;
}
```

```

}

getSelectedCategories(): string {
  const categories = this.activeFilters['category'];
  if (!categories || categories.length === 0) return 'None';
  return categories.map((c: FilterOption) => c.label).join(', ');
}

getSelectedPriceRanges(): string {
  const prices = this.activeFilters['price'];
  if (!prices || prices.length === 0) return 'None';
  return prices.map((p: FilterOption) => p.label).join(', ');
}

```

Multi-Selection Mode

Advanced multi-selection capabilities with grouped filters and clear all functionality.

```

<aava-filter
  size="md"
  title="Filter Users"
  [filterGroups]="filterGroups"
  [showBadge]="true"
  [showApplyButton]="true"
  (filterChange)="onFilterChange($event)"
  (clearAll)="onClearAllFilters()"
  (apply)="onApplyFilters($event)"
></aava-filter>

<table>
  <thead>
    <tr>
      <th>Name</th>
      <th>Status</th>
      <th>Role</th>
    
```

```

<th>Department</th>
<th>Experience</th>
</tr>
</thead>
<tbody>
<tr *ngFor="let user of filteredUsers">
<td>{{ user.name }}</td>
<td>
<span class="status-badge" [ngClass]="'status-' + user.status">
{{ user.status }}
</span>
</td>
<td>{{ user.role }}</td>
<td>{{ user.department }}</td>
<td>{{ user.experience }}</td>
</tr>
</tbody>
</table>
---
filterGroups = [
{
id: 'status',
title: 'Status',
options: [
{ id: 'active', label: 'Active', value: 'active' },
{ id: 'inactive', label: 'Inactive', value: 'inactive' },
{ id: 'pending', label: 'Pending', value: 'pending' },
{ id: 'archived', label: 'Archived', value: 'archived' },

```

```
],
},
{
  id: 'role',
  title: 'Role',
  options: [
    { id: 'admin', label: 'Admin', value: 'admin' },
    { id: 'user', label: 'User', value: 'user' },
    { id: 'moderator', label: 'Moderator', value: 'moderator' },
    { id: 'guest', label: 'Guest', value: 'guest' },
  ],
},
{
  id: 'department',
  title: 'Department',
  options: [
    { id: 'it', label: 'IT', value: 'IT' },
    { id: 'hr', label: 'HR', value: 'HR' },
    { id: 'finance', label: 'Finance', value: 'Finance' },
    { id: 'marketing', label: 'Marketing', value: 'Marketing' },
    { id: 'sales', label: 'Sales', value: 'Sales' },
  ],
},
{
  id: 'experience',
  title: 'Experience Level',
  options: [
    { id: 'junior', label: 'Junior (0-2 years)', value: 'junior' },
    { id: 'mid', label: 'Mid-level (3-5 years)', value: 'mid' },
  ]
}
```

```
{ id: 'senior', label: 'Senior (6-10 years)', value: 'senior' },  
],  
},  
];  
  
users = [  
{  
id: 1,  
name: 'John Doe',  
status: 'active',  
role: 'admin',  
department: 'IT',  
experience: 'senior',  
},  
{  
id: 2,  
name: 'Jane Smith',  
status: 'active',  
role: 'user',  
department: 'HR',  
experience: 'mid',  
},  
{  
id: 3,  
name: 'Bob Johnson',  
status: 'inactive',  
role: 'moderator',  
department: 'Finance',  
experience: 'senior',  
},
```

```
{  
  id: 4,  
  name: 'Alice Brown',  
  status: 'active',  
  role: 'user',  
  department: 'Marketing',  
  experience: 'junior',  
},  
{  
  id: 5,  
  name: 'Charlie Wilson',  
  status: 'pending',  
  role: 'user',  
  department: 'IT',  
  experience: 'mid',  
},  
{  
  id: 6,  
  name: 'Diana Davis',  
  status: 'active',  
  role: 'admin',  
  department: 'HR',  
  experience: 'expert',  
},  
{  
  id: 7,  
  name: 'Evan Miller',  
  status: 'archived',  
  role: 'guest',
```

```
        department: 'Sales',
        experience: 'junior',
    },
{
    id: 8,
    name: 'Fiona Garcia',
    status: 'active',
    role: 'moderator',
    department: 'IT',
    experience: 'senior',
},
];
filteredUsers = [...this.users];
activeFilters: Record<string, FilterOption[]> = {};
onFilterChange(filters: Record<string, FilterOption[]>): void {
    console.log('Filter changed:', filters);
    this.activeFilters = filters;
    this.applyFilters();
}
onClearAllFilters(): void {
    console.log('All filters cleared');
    this.activeFilters = {};
    this.filteredUsers = [...this.users];
}
onApplyFilters(filters: Record<string, FilterOption[]>): void {
    console.log('Filters applied:', filters);
    this.activeFilters = filters;
    this.applyFilters();
}
```

```
private applyFilters(): void {
  this.filteredUsers = this.users.filter((user) => {
    // Status filter
    if (
      this.activeFilters['status'] &&
      this.activeFilters['status'].length > 0
    ) {
      const statusMatch = this.activeFilters['status'].some(
        (f: FilterOption) => f.value === user.status
      );
      if (!statusMatch) return false;
    }
    // Role filter
    if (this.activeFilters['role'] && this.activeFilters['role'].length > 0) {
      const roleMatch = this.activeFilters['role'].some(
        (f: FilterOption) => f.value === user.role
      );
      if (!roleMatch) return false;
    }
    // Department filter
    if (
      this.activeFilters['department'] &&
      this.activeFilters['department'].length > 0
    ) {
      const deptMatch = this.activeFilters['department'].some(
        (f: FilterOption) => f.value === user.department
      );
      if (!deptMatch) return false;
    }
  }
}
```

```
// Experience filter

if (
  this.activeFilters['experience'] &&
  this.activeFilters['experience'].length > 0
) {
  const expMatch = this.activeFilters['experience'].some(
    (f: FilterOption) => f.value === user.experience
  );
  if (!expMatch) return false;
}

return true;
});

}

getActiveFiltersCount(): number {
  let count = 0;

  Object.values(this.activeFilters).forEach((group) => {
    if (Array.isArray(group)) {
      count += group.length;
    }
  });
  return count;
}

getFilterSummary(): string {
  const summaries: string[] = [];

  if (this.activeFilters['status']?.length) {
    summaries.push(` ${this.activeFilters['status'].length} status(es)`);
  }

  if (this.activeFilters['role']?.length) {
    summaries.push(` ${this.activeFilters['role'].length} role(s)`);
  }
}
```

```

}

if (this.activeFilters['department']?.length) {
  summaries.push(
    `${this.activeFilters['department'].length} department(s)`
  );
}

if (this.activeFilters['experience']?.length) {
  summaries.push(
    `${this.activeFilters['experience'].length} experience level(s)`
  );
}

return summaries.length > 0 ? summaries.join(', ') : 'No filters applied';
}

```

Multi-Selection Features

- Grouped Filters: Organize filters into logical groups with titles
- Multi-Select Groups: Support for both single and multi-selection within groups
- Clear All: Bulk clear functionality for all selected filters
- Active Count Badge: Visual indicator of active filter count
- Apply Button: Optional apply button for controlled filter application

Single-Selection Mode

Streamlined single-selection interface optimized for quick filtering decisions.

```

<aava-filter
  [singleSelection]="true"
  title="Sort By"
  [filterGroups]="sortFilterGroups"
  [showClearAll]="true"
  [showApplyButton]="false"
  size="md"

```

```

(singleSelectionChange)="onSortChange($event)"

></aava-filter>

<div class="product-card" *ngFor="let product of sortedProducts; let i = index">
<span class="order-number">#{{ i + 1 }}</span>
<h4>{{ product.name }}</h4>
<p>Category: {{ product.category }}</p>
<p class="price">${{ product.price }}</p>
<p class="date">{{ product.date | date : "shortDate" }}</p>
</div>

---

sortFilterGroups = [
{
  id: "sortBy",
  title: "Sort By",
  options: [
    { id: "name-asc", label: "Name (A-Z)", value: "name_asc" },
    { id: "name-desc", label: "Name (Z-A)", value: "name_desc" },
    { id: "price-low", label: "Price (Low to High)", value: "price_asc" },
    { id: "price-high", label: "Price (High to Low)", value: "price_desc" },
    { id: "newest", label: "Newest First", value: "date_desc" },
    { id: "oldest", label: "Oldest First", value: "date_asc" },
  ],
},
];
products = [
{
  id: 1,
  name: 'Laptop',
  category: 'electronics',

```

```
    price: 999,  
    date: '2024-01-15',  
},  
{  
    id: 2,  
    name: 'Smartphone',  
    category: 'electronics',  
    price: 699,  
    date: '2024-01-10',  
},  
{  
    id: 3,  
    name: 'T-Shirt',  
    category: 'clothing',  
    price: 25,  
    date: '2024-01-20',  
},  
{  
    id: 4,  
    name: 'Jeans',  
    category: 'clothing',  
    price: 45,  
    date: '2024-01-18',  
},  
{ id: 5, name: 'Novel', category: 'books', price: 15, date: '2024-01-12' },  
{  
    id: 6,  
    name: 'Cookbook',  
    category: 'books',
```

```
    price: 35,  
    date: '2024-01-14',  
},  
{  
    id: 7,  
    name: 'Coffee Table',  
    category: 'home',  
    price: 299,  
    date: '2024-01-16',  
},  
{  
    id: 8,  
    name: 'Garden Chair',  
    category: 'home',  
    price: 89,  
    date: '2024-01-19',  
},  
{  
    id: 9,  
    name: 'Running Shoes',  
    category: 'sports',  
    price: 120,  
    date: '2024-01-17',  
},  
{  
    id: 10,  
    name: 'Yoga Mat',  
    category: 'sports',  
    price: 45,
```

```
date: '2024-01-13',  
},  
];  
  
sortedProducts = [...this.products];  
  
selectedSort: FilterOption | null = null;  
  
selectedCategory: FilterOption | null = null;  
  
selectedStatus: FilterOption | null = null;  
  
onSortChange(selection: Record<string, FilterOption | null>): void {  
  const sortOption = selection['sortBy'];  
  
  this.selectedSort = sortOption;  
  
  console.log('Sort changed:', sortOption);  
  
  if (sortOption) {  
  
    this.applySorting(sortOption.value);  
  }  
}  
  
private applySorting(sortBy: string): void {  
  this.sortedProducts = [...this.products];  
  
  switch (sortBy) {  
  
    case 'name_asc':  
  
      this.sortedProducts.sort((a, b) => a.name.localeCompare(b.name));  
  
      break;  
  
    case 'name_desc':  
  
      this.sortedProducts.sort((a, b) => b.name.localeCompare(a.name));  
  
      break;  
  
    case 'price_asc':  
  
      this.sortedProducts.sort((a, b) => a.price - b.price);  
  
      break;  
  
    case 'price_desc':  
  
      this.sortedProducts.sort((a, b) => b.price - a.price);  
  }
```

```

break;

case 'date_desc':
  this.sortedProducts.sort(
    (a, b) => new Date(b.date).getTime() - new Date(a.date).getTime()
  );
  break;
case 'date_asc':
  this.sortedProducts.sort(
    (a, b) => new Date(a.date).getTime() - new Date(b.date).getTime()
  );
  break;
}
}

getSelectedSortLabel(): string {
  return this.selectedSort ? this.selectedSort.label : 'None';
}

getSelectedCategoryLabel(): string {
  return this.selectedCategory ? this.selectedCategory.label : 'None';
}

getSelectedStatusLabel(): string {
  return this.selectedStatus ? this.selectedStatus.label : 'None';
}

```

Single-Selection Features

- Radio-like Behavior: Only one option can be selected per group
- Auto-close: Panel automatically closes after selection
- Visual Feedback: Clear selection indicators with check icons
- Simplified Interface: No checkboxes, clean click-to-select interaction

Configuration

Flexible configuration and sizing options for different layout requirements.

[Error extracting code]

Features

Selection Modes

- Multi-Selection: Default mode with checkbox-based selection
- Single-Selection: Radio-like behavior with auto-close
- Group-Level Control: Individual groups can override selection behavior
- Mixed Modes: Support for different selection types per group

User Experience

- Visual Feedback: Active state indicators and selection badges
- Keyboard Navigation: Full keyboard support for accessibility
- Responsive Design: Adapts to different screen sizes
- Smooth Animations: CSS transitions for panel open/close
- State Persistence: Maintains selection state during interactions

Filter Management

- Grouped Organization: Logical grouping of related filter options
- Clear All: Bulk clear functionality for all filters
- Apply Control: Optional apply button for controlled submission
- Active Count: Real-time display of active filter count
- Position Flexibility: Left or right-aligned panels

API Reference

Inputs

[TABLE]

Property | Type | Default | Description

size | FilterSize | 'md' | Size variant (sm, md, lg, xl)

title | string | 'Filter' | Title displayed on the filter button

filterGroups | FilterGroup[] | [] | Array of filter groups with options

showClearAll | boolean | true | Whether to show clear all button
showApplyButton | boolean | false | Whether to show apply button
isOpen | boolean | false | Whether the filter panel is open
position | 'left' | 'right' | 'left' | Position of the filter panel
maxHeight | string | '400px' | Maximum height of the filter panel
width | string | 'auto' | Width of the filter panel
disabled | boolean | false | Whether the filter is disabled
class | string | '' | Additional CSS classes
singleSelection | boolean | false | Enable single-selection mode
showBadge | boolean | true | Whether to show active filter count badge
[/TABLE]

Outputs

[TABLE]

Event	Type	Description
filterChange	EventEmitter<{ [groupId: string]: FilterOption[] }>	Emitted when filters change (multi-select)
clearAll	EventEmitter<void>	Emitted when clear all is clicked
apply	EventEmitter<{ [groupId: string]: FilterOption[] }>	Emitted when apply button is clicked
toggleFilter	EventEmitter<boolean>	Emitted when filter panel is toggled
singleSelectionChange	EventEmitter<{ [groupId: string]: FilterOption null }>	Emitted when selection changes (single-select)

[/TABLE]

Methods

[TABLE]

Method	Parameters	Return	Description
toggleFilterPanel()	None	void	Toggle the filter panel open/close
onOptionChange()	groupId: string, option: FilterOption, isChecked: boolean	void	Handle option selection change
onSingleOptionClick()	groupId: string, option: FilterOption	void	Handle single-selection option click
clearAllFilters()	None	void	Clear all selected filters

applyFilters() | None | void | Apply current filter selection
getActiveFiltersCount() | None | number | Get count of active filters
getSizeClasses() | None | string | Get CSS classes for current size
getCheckboxSize() | None | 'sm' | 'md' | 'lg' | Get appropriate checkbox size for current size
getIconSize() | None | number | Get appropriate icon size for current size
isOptionSelected() | groupId: string, option: FilterOption | boolean | Check if option is selected

[/TABLE]

Properties

[TABLE]

Property | Type | Description

selectedFilters | { [groupId: string]: FilterOption[] } | Current multi-selection state (private)

singleSelectedFilters | { [groupId: string]: FilterOption | null } | Current single-selection state (private)

[/TABLE]

Interfaces

FilterOption

interface

FilterOption

{

id

:

string

|

number

;

label

:

string

;

```
value
```

```
:
```

```
any
```

```
;
```

```
selected
```

```
?
```

```
:
```

```
boolean
```

```
;
```

```
}
```

FilterGroup

```
interface
```

```
FilterGroup
```

```
{
```

```
id
```

```
:
```

```
string
```

```
;
```

```
title
```

```
:
```

```
string
```

```
;
```

```
options
```

```
:
```

```
FilterOption
```

```
[
```

```
]
```

```
;
```

```
multiSelect
```

```
?  
:  
boolean  
;  
}
```

FilterSize

type

FilterSize

=

"sm"

|

"md"

|

"lg"

|

"xlg"

;

Design Tokens & Theming

AAVA Play Filter 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 Filter

[TABLE]

Token | Purpose | Default Value

--filter-background-primary | Primary background color | Theme-based

--filter-background-hover | Hover background color | Theme-based

--filter-border | Border style | Theme-based

--filter-border-hover | Hover border color | Theme-based

--filter-border-focus | Focus border color | Theme-based

--filter-border-radius | Border radius | Theme-based
--filter-text-color | Text color | Theme-based
--filter-text-active | Active text color | Theme-based
--filter-disabled-opacity | Disabled state opacity | Theme-based

[/TABLE]

[TABLE]

Token | Purpose | Default Value
--filter-badge-background | Badge background color | Theme-based
--filter-badge-text | Badge text color | Theme-based
--filter-badge-border-radius | Badge border radius | Theme-based

[/TABLE]

[TABLE]

Token | Purpose | Default Value
--filter-panel-background | Panel background color | Theme-based
--filter-panel-border | Panel border style | Theme-based
--filter-panel-shadow | Panel shadow | Theme-based
--filter-header-background | Header background color | Theme-based
--filter-header-border | Header border style | Theme-based
--filter-header-padding | Header padding | Theme-based

[/TABLE]

Token Override Example

```
/* Custom filter theming */
```

```
.my-custom-filter
```

```
{
```

```
--filter-background-primary
```

```
:
```

f8fafc

```
;
```

```
--filter-border
```

```
:  
1  
px  
solid  
e2e8f0  
;  
--filter-border-radius  
:  
8  
px  
;  
--filter-panel-shadow  

```

```
)  
;  
}  
.my-compact-filter  
{  
--filter-header-padding  
:  
12  
px  
16  
px  

```

Best Practices

Design Guidelines

- Logical Grouping: Organize filters into meaningful, related groups
- Clear Labels: Use descriptive, concise labels for filter options
- Consistent Sizing: Choose appropriate size variants for your interface context
- Positioning: Consider layout constraints when choosing left/right positioning
- Badge Usage: Show active filter count for better user awareness

Accessibility

- Keyboard Navigation: Ensure full keyboard support for all interactions
- Screen Reader Support: Provide clear labels and state announcements

- Focus Management: Proper focus handling when panel opens/closes
- Color Contrast: Maintain sufficient contrast for all text and interactive elements
- ARIA Labels: Use appropriate ARIA attributes for filter groups and options

Performance

- OnPush Strategy: Component uses OnPush change detection for optimal performance
- Efficient Rendering: Minimal re-renders during filter state changes
- Memory Management: Proper cleanup of event listeners and references
- Large Datasets: Consider pagination or virtualization for very large filter groups

User Experience

- Selection Modes: Choose between single and multi-selection based on use case
- Auto-close: Use auto-close for single-selection to streamline workflow
- Apply Button: Show apply button when immediate application isn't desired
- Clear All: Always provide clear all functionality for better usability
- Visual Feedback: Clear indication of active filters and selection states

Integration

- State Management: Integrate with your application's state management system
- Event Handling: Use the comprehensive event system for all filter interactions
- Form Integration: Coordinate with form validation and submission logic
- Data Binding: Properly bind filter data to your data source
- Responsive Design: Ensure filter works well on mobile and desktop devices

Responsive Behavior

Mobile Adaptations

The filter component automatically adapts to mobile screens:

- Touch Optimization: Optimized touch targets for mobile interaction
- Responsive Sizing: Appropriate sizing for mobile viewports
- Mobile Positioning: Smart positioning to avoid viewport edges
- Touch Gestures: Support for touch-based interactions

Breakpoint Behavior

- Desktop (>768px): Full filter interface with all features
- Mobile ($\leq 768\text{px}$): Compact layout with optimized spacing
- Panel Sizing: Responsive panel width and height
- Icon Sizing: Appropriate icon sizes for different screens

Content Considerations

- Group Layout: Filter groups adapt to different screen widths
- Option Display: Options maintain readability on small screens
- Button Sizing: Adequate touch target sizes for mobile
- Text Scaling: Appropriate text sizes for different screen sizes

ACCORDION

A responsive collapsible content component for organizing and displaying expandable sections. Features smooth height-based animations, flexible icon positioning, controlled/uncontrolled modes, and comprehensive accessibility support for enhanced user experience.

How to use

```
import {  
  AavaAccordionComponent  
}  
from "@aava/play-core"  
;
```

Basic Usage

Simple accordion implementations with expandable content sections.

```
<aava-accordion type="default" iconClosed="chevron-right" iconOpen="chevron-up">  
  <span header>Frequently Asked Questions</span>  
  <div content>  
    <p>  
      Here are the most common questions and answers about our product and  
      services.  
    </p>  
    <ul>  
      <li>How do I get started?</li>  
      <li>What are the pricing plans?</li>  
      <li>How can I contact support?</li>  
    </ul>  
  </div>  
</aava-accordion>
```

Icons

Lucide icon integration with customizable states and positioning.

```
<aava-accordion  
*ngFor="let config of iconConfigs"  
[iconClosed]="config.closed"  
[iconOpen]="config.open"  
[iconPosition]="config.position"  
(click)="onAccordionToggle($event)"  
[expanded]="config.expanded ?? false"  
>  
<span header>{{ config.name }}</span>  
<div content>  
<p>{{ config.description }}</p>  
<p>  
This accordion uses <strong>{{ config.closed }}</strong> for closed state  
and <strong>{{ config.open }}</strong> for open state, positioned on the  
<strong>{{ config.position }}</strong>.  
</p>  
</div>  
</aava-accordion>  
---  
iconConfigs = [  
{  
name: "Plus/Minus",  
closed: "plus",  
open: "minus",  
position: "right" as const,  
description: "Clear expand/collapse indication with plus and minus icons",  
},
```

```
{
  name: "Chevron Right/Up",
  closed: "chevron-right",
  open: "chevron-up",
  position: "left" as const,
  description: "Traditional chevron icons with left positioning",
  expanded: false,
},
];
onAccordionToggle(event: Event): void {
  console.log('Accordion toggled:', event);
}
```

Icon Features

- State Icons: Different icons for expanded and collapsed states
- Title Icons: Static icons for content categorization
- Positioning: Left or right icon placement options

Positions

Flexible icon positioning for optimal layout integration.

```
<aava-accordion
*ngFor="let example of positionExamples"
[iconPosition]="example.position"
[iconClosed]="example.iconClosed"
[iconOpen]="example.iconOpen"
(click)="onAccordionToggle($event)"
>
<span header>{{ example.title }}</span>
<div content>
<p>
```

```
<strong>{{ example.description }}</strong>
</p>
<p>
This accordion demonstrates the

<code>{{ example.position }}</code> icon positioning. The icons are
positioned on the {{ example.position }} side of the header, providing a
different visual emphasis compared to the opposite position.

</p>
</div>
</aava-accordion>

---
positionExamples = [
{
  position: "left" as const,
  title: "Left Position (Default)",
  description: "Icons positioned on the left side of the header",
  iconClosed: "chevron-right",
  iconOpen: "chevron-down",
},
{
  position: "right" as const,
  title: "Right Position",
  description:
    "Icons positioned on the right side for different visual emphasis",
  iconClosed: "chevron-right",
  iconOpen: "chevron-up",
},
];

```

Position Options

- Left Icons: Icons on the left side of the header (default)
- Right Icons: Icons on the right side for different visual emphasis

Accessibility

Built-in accessibility features ensuring inclusive user experience.

Accessibility Features

- Keyboard Navigation: Tab navigation and Enter/Space activation
- ARIA Support: Proper button role and expanded state announcements
- Screen Reader: Descriptive content and state information
- Focus Management: Clear visual focus indicators
- High Contrast: Enhanced visibility in high contrast modes
- Semantic HTML: Proper heading hierarchy and content structure

API Reference

Inputs

[TABLE]

Property	Type	Default	Description
expanded	boolean	false	Whether the accordion is currently expanded
animation	boolean	true	Whether to enable smooth expand/collapse animations
controlled	boolean	false	Whether expansion is controlled externally
iconClosed	string	"	Lucide icon name for collapsed state
iconOpen	string	"	Lucide icon name for expanded state
titleIcon	string	"	Static icon for title area (titleIcon type)
iconPosition	'left' 'right' 'left'	'left'	Position of expand/collapse icons
type	'default' 'titleIcon' 'default'	'default'	Layout type of the accordion

[/TABLE]

Properties

[TABLE]

Property	Type	Description
contentHeight	number	Calculated height of content for animations

accordionClasses | object | Computed CSS classes for styling

[/TABLE]

Methods

[TABLE]

Method | Parameters | Return Type | Description

toggleExpand() | - | void | Toggle expanded state (if not controlled)

onAccordionKeydown() | event: KeyboardEvent | void | Handle keyboard activation

ngAfterViewInit() | - | void | Calculate content height for animations

[/TABLE]

Content Projection

[TABLE]

Selector | Description

[header] | Content projected into the accordion header

[content] | Content projected into the expandable body

[/TABLE]

CSS Custom Properties

[TABLE]

Property | Default | Description

--accordion-light-background | Theme-based | Background color for light theme

--accordion-dark-header-background | Theme-based | Header background for dark theme

--accordion-dark-content-text | Theme-based | Content text color for dark theme

--accordion-content-font | Theme-based | Font size for content text

--accordion-font-family | Inter | Font family for accordion text

--accordion-font-weight | Theme-based | Font weight for content

--color-border-focus | Theme-based | Border color for focus states

[/TABLE]

Best Practices

Design Guidelines

- Use clear, descriptive headers that indicate the content
- Group related information logically within accordion sections
- Consider default expanded states for important content
- Use consistent icon patterns across related accordions
- Provide adequate spacing between multiple accordions

Accessibility

- Always provide meaningful header text for screen readers
- Use semantic heading levels for accordion headers when appropriate
- Ensure sufficient color contrast for all text content
- Test keyboard navigation thoroughly
- Consider announcing content changes to screen readers

Performance

- Use OnPush change detection strategy for optimal performance
- Avoid frequent expansion state changes that trigger animations
- Consider virtualization for large lists of accordions
- Cache content height calculations when possible

Technical Notes

Content Projection

Angular content projection enables flexible layouts:

- [header]selector for accordion headers
- [content]selector for expandable content
- Support for complex nested content and components

State Management

- Uncontrolled: Component manages its own expanded state
- Controlled: Parent component controls expansion via controlled property
- Keyboard and click events respect controlled mode settings

Icon Integration

Uses Lucide Angular for consistent iconography:

- Icons automatically rotate 180 degrees when expanded
- Different icons for open/closed states
- Static title icons for content categorization

TIMELINE

The Timeline component is a powerful and flexible way to display chronological events, project milestones, or any sequential information. It provides multiple orientations, status indicators, and extensive customization options while maintaining accessibility and responsive design.

The

```
<aava-timeline>
```

component supports both vertical and horizontal layouts, various text alignment patterns, multiple sizes, and status-based visual indicators. It's perfect for project management, user onboarding flows, historical displays, and any scenario requiring chronological representation.

How to use

```
import  
{  
AavaTimeline  
,  
TimelineEvent  
}  
from  
"@aava/play-core"  
;
```

Basic Timeline

A simple vertical timeline with default settings.

```
<aava-timeline  
[events]="basicEvents"  
[size]="'large'"  
[orientation]="'vertical'"  
[textAlign]="'zigzag-left'"  
[iconCircleSize]="'40px'"  
[iconCircleBgColor]="#fff"  
[iconCircleBorderColor]="#ccc"  
></aava-timeline>
```

```
basicEvents: TimelineEvent[] = [
{
  time: '10:00 AM',
  title: 'Project Kickoff Meeting',
  description: 'Initial project planning and team alignment session',
  iconName: 'circle-check',
  iconColor: '#E91E63',
  iconSize: '24px',
  year: '2025',
  status: 'completed',
},
{
  time: '2:00 PM',
  title: 'Development Phase Started',
  description: 'Begin coding and implementation of core features',
  iconName: 'loader',
  iconColor: '#E91E63',
  iconSize: '20px',
  year: '2025',
  status: 'current',
},
{
  time: '4:30 PM',
  title: 'First Milestone Completed',
  description: 'Successfully completed the initial development phase',
  iconName: 'check-circle',
  iconColor: '#E91E63',
  iconSize: '16px',
}
```

```
year: '2025',  
status: 'pending',  
,  
];  
  
selectedEvent: TimelineEvent | null = null;  
  
onEventSelect(event: TimelineEvent): void {  
  console.log('Timeline event selected:', event);  
  this.selectedEvent = event;  
}
```

Text Alignment Variants

Different text alignment patterns for visual variety.

```
<aava-timeline  
[events]="alignmentEvents"  
[size]="'large'"  
[orientation]="'vertical'"  
[textAlign]="'zigzag-left'"  
[iconCircleSize]="'40px'"  
[iconCircleBgColor]="#fff"  
[iconCircleBorderColor]="#ccc"  
></aava-timeline>  
  
<aava-timeline  
[events]="alignmentEvents"  
[size]="'large'"  
[orientation]="'vertical'"  
[textAlign]="'zigzag-right'"  
[iconCircleSize]="'40px'"  
[iconCircleBgColor]="#fff"  
[iconCircleBorderColor]="#ccc"  
></aava-timeline>
```

```
<aava-timeline
[events]="alignmentEvents"
[size]="'large'"
[orientation]="'vertical'"
[textAlign]="'zig-zag'"
[iconCircleSize]="'40px'"
[iconCircleBgColor]="#fff"
[iconCircleBorderColor]="#ccc"
></aava-timeline>

---
alignmentEvents: TimelineEvent[] = [
{
  time: '9:00 AM',
  title: 'Morning Standup',
  description: 'Daily team synchronization meeting',
  iconName: 'circle-check',
  iconColor: '#E91E63',
  iconSize: '24px',
  year: '2025',
  status: 'completed',
},
{
  time: '11:30 AM',
  title: 'Code Review',
  description: 'Peer review of new feature implementation',
  iconName: 'check',
  iconColor: '#E91E63',
  iconSize: '20px',
  year: '2025',
}
```

```
status: 'current',
},
{
time: '2:00 PM',
title: 'Testing Phase',
description: 'Quality assurance and bug testing',
iconName: 'circle-check',
iconColor: '#E91E63',
iconSize: '16px',
year: '2025',
status: 'pending',
},
{
time: '4:30 PM',
title: 'Deployment',
description: 'Production deployment and monitoring',
iconName: 'rocket',
iconColor: '#E91E63',
iconSize: '18px',
year: '2025',
status: 'unreached',
},
];
selectedEvent: TimelineEvent | null = null;
onEventSelect(event: TimelineEvent): void {
  console.log('Timeline event selected:', event);
  this.selectedEvent = event;
}
```

Orientations

Vertical and horizontal timeline layouts.

[Error extracting code]

Size Variants

Multiple size options for different use cases.

[Error extracting code]

API Reference

Input Properties

[TABLE]

Property | Type | Default | Description

events | TimelineEvent[] | [] | Array of timeline events to display

orientation | 'vertical' | 'horizontal' | 'vertical' | Timeline layout direction

size | 'x-small' | 'small' | 'medium' | 'large' | 'large' | Overall size of the timeline

sortOrder | 'ascending' | 'descending' | 'ascending' | Order to sort events

textAlign | 'zig-zag' | 'zigzag-left' | 'zigzag-right' | 'zigzag-up' | 'zigzag-down' | 'zigzag-left' | Text alignment pattern

iconCircleBgColor | string | '#fff' | Background color for icon circles

iconCircleBorderColor | string | '#ccc' | Border color for icon circles

iconCircleSize | string | '40px' | Size of icon circles

disabled | boolean | false | Disables the entire timeline

customSort | (a: TimelineEvent, b: TimelineEvent) => number | undefined | Custom sorting function

contentTemplate | any | undefined | Template for custom content projection

[/TABLE]

TimelineEvent Interface

interface

TimelineEvent

{

time

?

```
:  
string  
;  
  
// Time of the event  
  
text  
?  
:  
string  
;  
  
// Short text description  
  
iconName  
?  
:  
string  
;  
  
// Icon name from icon library  
  
iconColor  
?  
:  
string  
;  
  
// Custom icon color  
  
iconSize  
?  
:  
string  
;  
  
// Custom icon size  
  
imageUrl
```

```
?  
:  
string  
;  
  
// Image URL for custom images  
  
imageSize  
?  
:  
string  
;  
  
// Custom image size  
  
year  
?  
:  
string  
;  
  
// Year of the event  
  
title  
?  
:  
string  
;  
  
// Event title  
  
description  
?  
:  
string  
;  
  
// Detailed description
```

```
status
?
:
"completed"
|
"current"
|
"pending"
|
"unreached"
;

customClass
?
:
string
;

// Additional CSS classes

data
?
:
any
;

// Custom data for advanced use cases
}
```

Features

Multiple Orientations

- Vertical: Traditional top-to-bottom timeline layout
- Horizontal: Left-to-right layout for wide screens

Text Alignment Patterns

- zig-zag: Alternating left/right alignment for vertical, up/down for horizontal
- zigzag-left: All text aligned to the left
- zigzag-right: All text aligned to the right
- zigzag-up: All text aligned upward (horizontal only)
- zigzag-down: All text aligned downward (horizontal only)

Status System

- completed: Finished events with success styling
- current: Currently active events with highlight styling
- pending: Upcoming events with neutral styling
- unreached: Future events with disabled styling

Size Variants

- x-small: Compact size for dense interfaces
- small: Small size for limited space
- medium: Standard size for most use cases
- large: Large size for prominent displays

Customization Options

- Custom icon and image support
- Flexible color theming
- Custom sorting functions
- Content projection for advanced layouts
- Responsive design with automatic spacing

Best Practices

Content Organization

- Use clear, descriptive titles for each event
- Keep descriptions concise but informative
- Use consistent date/time formats
- Group related events logically

Visual Hierarchy

- Choose appropriate sizes based on content importance
- Use status indicators consistently across events
- Maintain visual balance with text alignment patterns
- Consider screen size when choosing orientation

Accessibility

- Provide meaningful alt text for images
- Use semantic status indicators
- Ensure sufficient color contrast
- Support keyboard navigation

Performance

- Limit the number of events for optimal performance
- Use custom sorting functions sparingly
- Consider lazy loading for large timelines
- Optimize images and icons

Examples

Project Timeline

```
const
```

```
projectTimeline
```

```
:
```

```
TimelineEvent
```

```
[
```

```
]
```

```
=
```

```
[
```

```
{
```

```
year
```

```
:
```

```
"2023 Q1"

,
title
:

"Planning Phase"

,
description
:

"Requirements gathering and project planning"

,
status
:

"completed"

,
iconName
:

"check-circle"

,
}

,
{

year
:

"2023 Q2"

,
title
:

"Development"
,
```

```
description
:
"Core feature development"
,
status
:
"current"
,
iconName
:
"code"
,
}
,
{
year
:
"2023 Q3"
,
title
:
"Testing"
,
description
:
"Quality assurance and testing"
,
status
:
```

```
"pending"
,
iconName
:
"bug"
,
}
,
{
year
:
"2023 Q4"
,
title
:
"Launch"
,
description
:
"Production deployment"
,
status
:
"unreached"
,
iconName
:
"rocket"
,
```

```
}
```

```
,
```

```
]
```

```
;
```

User Journey Timeline

```
const
```

```
userJourney
```

```
:
```

```
TimelineEvent
```

```
[
```

```
]
```

```
=
```

```
[
```

```
{
```

```
time
```

```
:
```

```
"0 min"
```

```
,
```

```
title
```

```
:
```

```
"Welcome"
```

```
,
```

```
description
```

```
:
```

```
"User lands on homepage"
```

```
,
```

```
status
```

```
:
```

```
"completed"
```

,

iconName

:

"home"

,

}

,

{

time

:

"2 min"

,

title

:

"Browse"

,

description

:

"User explores content"

,

status

:

"current"

,

iconName

:

"search"

,

}

```
,
```

```
{
```

```
  time
```

```
:
```

```
  "5 min"
```

```
,
```

```
  title
```

```
:
```

```
  "Engage"
```

```
,
```

```
  description
```

```
:
```

```
  "User interacts with features"
```

```
,
```

```
  status
```

```
:
```

```
  "pending"
```

```
,
```

```
  iconName
```

```
:
```

```
  "heart"
```

```
,
```

```
}
```

```
,
```

```
]
```

```
;
```

Styling

The timeline component uses CSS custom properties for theming and supports extensive customization through:

- Icon circle colors and sizes
- Status-based color schemes
- Responsive spacing adjustments
- Custom CSS classes for advanced styling

TREEVIEW

A hierarchical tree view component that provides an intuitive way to display and navigate nested data structures. Built with accessibility in mind, it supports expandable/collapsible nodes, customizable icons, multiple size variants, and comprehensive keyboard navigation for building file browsers, navigation menus, and organizational charts.

How to use

```
import {  
    AavaTreeviewComponent,  
    TreeNode  
} from "@aava/play-core";
```

Note

: The TreeView component is standalone and includes all necessary dependencies for common modules and Lucide icons.

Basic Usage

Simple tree view with expandable nodes and basic selection.

```
<div *ngFor="let config of treeConfigs" class="tree-variant">  
    <aava-treeview  
        [nodes]="config.nodes"  
        [size]="config.size"  
        [iconPosition]="config.iconPosition"  
        (nodeSelect)="onNodeSelect(config, $event)">  
    </aava-treeview>  
</div>  
---
```

```
export interface TreeNode {  
  id?: string | number;  
  name: string;  
  icon?: string;  
  expanded?: boolean;  
  selected?: boolean;  
  level?: number;  
  children?: TreeNode[];  
}  
  
interface TreeviewConfig {  
  size: 'xs' | 'sm' | 'md' | 'lg' | 'xl';  
  iconPosition: 'left' | 'right';  
  nodes: TreeNode[];  
}  
  
treeConfigs: TreeviewConfig[] = [  
{  
  size: 'md',  
  iconPosition: 'left',  
  nodes: this.makeSampleTree(),  
},  
];  
  
private makeSampleTree(): TreeNode[] {  
  return [  
{  
  id: '1',  
  name: 'Engineering',  
  icon: 'folder',  
  expanded: false,  
  selected: false,
```

```
children: [
  { id: '1.1', name: 'Frontend', icon: 'folder', selected: false },
  { id: '1.2', name: 'Backend', icon: 'folder', selected: false },
],
},
{
id: '2',
name: 'Mobile',
icon: 'folder',
expanded: false,
selected: false,
children: [
  { id: '2.1', name: 'UI', icon: 'folder', selected: false },
  { id: '2.2', name: 'Sap', icon: 'folder', selected: false },
],
},
{
id: '3', name: 'Marketing', icon: 'folder', selected: false },
{ id: '4', name: 'Operations', icon: 'folder', selected: false },
];
}

onNodeSelect(config: TreeviewConfig, node: TreeNode) {
  console.log('Selected from', ' ', node);
  // update selection state
  config.nodes = this.updateTreeSelection(config.nodes, node);
}

private updateTreeSelection(
  nodes: TreeNode[],
  targetNode: TreeNode
): TreeNode[] {
```

```

if (!nodes) return [];

return nodes.map((n) => {
  const newNode: TreeNode = { ...n };

  if (newNode.children?.length) {
    newNode.children = this.updateTreeSelection(
      newNode.children,
      targetNode
    );
  }

  newNode.selected = newNode.id === targetNode.id;

  return newNode;
});
}

```

Features

Hierarchical Structure

- Nested Nodes: Support for unlimited nesting levels
- Expandable/Collapsible: Interactive nodes that can be expanded or collapsed
- Dynamic Indentation: Automatic indentation based on node level
- Recursive Rendering: Self-referential component for nested structures

Visual Customization

- Multiple Sizes: Five size variants (xs, sm, md, lg, xl)
- Icon Positioning: Left or right-aligned expand/collapse controls
- Custom Icons: Support for Lucide icons and folder states
- Responsive Design: Adapts to different screen sizes

User Interaction

- Node Selection: Click to select individual nodes
- Keyboard Navigation: Full keyboard support for accessibility
- Expand/Collapse: Click toggle controls or use arrow keys

- Hover States: Visual feedback for interactive elements

Accessibility

- ARIA Support: Proper ARIA attributes for screen readers
- Keyboard Navigation: Arrow keys, Enter, and Space for interaction
- Focus Management: Clear focus indicators and logical tab order
- Semantic Structure: Proper HTML semantics for tree navigation

API Reference

Inputs

[TABLE]

Property | Type | Default | Description

nodes | TreeNode[] | [] | Array of tree nodes to display

size | 'xs' | 'sm' | 'md' | 'lg' | 'xl' | 'md' | Size variant for the tree nodes

iconPosition | 'left' | 'right' | 'left' | Position of expand/collapse controls

level | number | 0 | Current nesting level (used internally)

[/TABLE]

Outputs

[TABLE]

Event | Type | Description

nodeSelect | EventEmitter<TreeNode> | Emitted when a node is selected

[/TABLE]

Methods

[TABLE]

Method | Parameters | Return | Description

toggleExpand() | node: TreeNode | void | Toggle the expanded state of a node

selectNode() | node: TreeNode | void | Select a node and emit selection event

calculateIndent() | level?: number | number | Calculate indentation for a given level

handleKeyDown() | event: KeyboardEvent, node: TreeNode | void | Handle keyboard navigation events

[/TABLE]

Interfaces

TreeNode

```
interface  
  TreeNode  
  {  
    id  
    ?  
    :  
    string  
  |  
    number  
  ;  
  // Unique identifier for the node  
  name  
  :  
  string  
  ;  
  // Display name for the node  
  icon  
  ?  
  :  
  string  
  ;  
  // Lucide icon name (optional)  
  expanded  
  ?  
  :  
  boolean  
  ;
```

```

// Whether the node is expanded
selected
?
:
boolean
;

// Whether the node is selected
level
?
:
number
;

// Nesting level (auto-calculated)
children
?
:
TreeNode
[
]
;

// Child nodes (optional)
}

```

Focus Management

- Each tree node is focusable with tabindex="0"
- Toggle controls have tabindex="-1" to prevent tab navigation
- Focus indicators provide clear visual feedback
- Logical tab order follows the tree structure

Design Tokens & Theming

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

Available Design Tokens for TreeView

[TABLE]

Token | Purpose | Default Value

Token	Purpose	Default Value
--tree-node-gap	Gap between node elements	Theme-based
--tree-node-height-xs	Extra small node height	Theme-based
--tree-node-height-sm	Small node height	Theme-based
--tree-node-height-md	Medium node height	Theme-based
--tree-node-height-lg	Large node height	Theme-based
--tree-node-height-xl	Extra large node height	Theme-based
--tree-node-font-weight-xl	Font weight for extra large	Theme-based
--tree-node-line-height-xs	Line height for extra small	Theme-based
--tree-node-line-height-medium	Line height for medium	Theme-based
--tree-node-line-height-lg	Line height for large	Theme-based
--tree-node-line-height-xl	Line height for extra large	Theme-based

[/TABLE]

[TABLE]

Token | Purpose | Default Value

Token	Purpose	Default Value
--tree-toggle-size-xs	Extra small toggle width	Theme-based
--tree-toggle-size-sm	Small toggle width	Theme-based
--tree-toggle-size-md	Medium toggle width	Theme-based
--tree-toggle-size-lg	Large toggle width	Theme-based
--tree-toggle-size-xl	Extra large toggle width	Theme-based

[/TABLE]

[TABLE]

Token | Purpose | Default Value

Token	Purpose	Default Value
--tree-icon-size-xs	Extra small icon size	Theme-based
--tree-icon-size-sm	Small icon size	Theme-based

--tree-icon-size-lg | Large icon size | Theme-based
--tree-icon-size-xl | Extra large icon size | Theme-based
[/TABLE]
[TABLE]
Token | Purpose | Default Value
--tree-label-font-family | Font family for labels | Theme-based
--tree-label-font-size-xs | Extra small font size | Theme-based
--tree-label-font-size-sm | Small font size | Theme-based
--tree-label-font-size-medium | Medium font size | Theme-based
--tree-label-font-size-lg | Large font size | Theme-based
--tree-label-font-size-xl | Extra large font size | Theme-based

[/TABLE]
[TABLE]
Token | Purpose | Default Value
--color-text-primary | Primary text color | Theme-based
--rgb-brand-disabled | Brand color for states | Theme-based
[/TABLE]

Token Override Example

/* Custom tree view theming */

```
.my-custom-tree
{
  --tree-node-gap
  :
  12
  px
  ;
  --tree-node-height-md
  :
```

```
40
px
;
--tree-label-font-size-medium
:
16
px
;
--tree-icon-size-lg
:
20
px
;
}
.my-compact-tree
{
--tree-node-height-md
:
32
px
;
--tree-label-font-size-medium
:
14
px
;
--tree-icon-size-lg
:
16
```

```
px
;
}

.my-spacious-tree
{
--tree-node-gap
:
16

px
;
--tree-node-height-md
:
48

px
;
--tree-label-font-size-medium
:
18

px
;
--tree-icon-size-lg
:
24

px
;
}
```

Best Practices

Design Guidelines

- Consistent Hierarchy: Use consistent indentation and visual cues
- Clear Labels: Ensure node names are descriptive and concise
- Appropriate Icons: Use meaningful icons that represent node types
- Size Selection: Choose size variants that match your content density
- Icon Positioning: Consider user expectations for expand/collapse controls

Accessibility

- Keyboard Navigation: Ensure all interactions work with keyboard
- Screen Reader Support: Provide clear labels and descriptions
- Focus Indicators: Maintain visible focus states
- ARIA Attributes: Use proper ARIA roles and properties
- Color Contrast: Ensure sufficient contrast for text and icons

Performance

- Lazy Loading: Consider lazy loading for large tree structures
- Virtual Scrolling: Implement virtual scrolling for very large trees
- Change Detection: Use OnPush strategy for better performance
- Memory Management: Clean up event listeners and references

User Experience

- Visual Feedback: Provide clear hover and selection states
- Smooth Animations: Use transitions for expand/collapse actions
- Consistent Behavior: Maintain predictable interaction patterns
- Error Handling: Gracefully handle invalid data structures

Integration

- Data Structure: Ensure your data follows the TreeNode interface
- Event Handling: Implement proper selection and expansion logic
- State Management: Coordinate tree state with your application
- Styling: Use design tokens for consistent theming

Responsive Behavior

Mobile Adaptations

The tree view component automatically adapts to mobile screens:

- Touch Optimization: Appropriate touch targets for mobile interaction
- Mobile Layout: Optimized spacing and sizing for small screens
- Gesture Support: Touch-friendly expand/collapse interactions
- Responsive Icons: Icon sizes that work well on mobile

Breakpoint Behavior

- Desktop (>768px): Full tree interface with all features
- Mobile ($\leq 768\text{px}$): Compact layout with optimized spacing
- Node Display: Responsive node sizing and spacing
- Icon Scaling: Appropriate icon sizes for different screens

Content Considerations

- Node Names: Node labels adapt to different screen widths
- Indentation: Appropriate indentation levels for mobile
- Icon Visibility: Icons remain visible and accessible
- Touch Targets: Adequate touch target sizes for mobile

Use Cases

File System Navigation

- File Browsers: Navigate through directory structures
- Document Management: Organize and browse documents
- Media Libraries: Browse photo and video collections
- Code Repositories: Navigate project file structures

Organizational Charts

- Company Structure: Display organizational hierarchy
- Team Management: Show team relationships and roles
- Project Structure: Organize project components
- Category Management: Display product or content categories

Navigation Systems

- Website Navigation: Site structure and menu systems
- Application Menus: App navigation and settings
- Breadcrumb Navigation: Hierarchical navigation paths
- Sitemap Display: Website structure visualization

Data Visualization

- Hierarchical Data: Display nested data relationships
- Taxonomy Systems: Show classification hierarchies
- Decision Trees: Visualize decision-making processes
- Workflow Diagrams: Display process flows and steps

CAROUSEL

BREADCRUMB

The

```
<aava-breadcrumbs>
```

provides a comprehensive navigation solution for displaying hierarchical navigation paths. It supports icons, collapsible behavior, multiple sizes, custom separators, and responsive design to enhance user navigation experience.

How to use

Import the component and provide an array of breadcrumb items with their navigation properties.

```
import {  
    AavaBreadcrumbsComponent  
} from "@aava/play-core";
```

Basic Usage

Simple breadcrumb navigation with text labels.

```
<aava-breadcrumbs [breadcrumbs]="simpleBreadcrumbs"></aava-breadcrumbs>  
---  
simpleBreadcrumbs: BreadcrumbItem[] = [  
    { label: 'Home', url: '/home', active: false },  
    { label: 'Category', url: '/category', active: false },  
    { label: 'Product', url: '/product', active: true },  
];
```

With Icons

Breadcrumbs with icons for enhanced visual representation.

```
<aava-breadcrumbs [breadcrumbs]="mixedBreadcrumbs"></aava-breadcrumbs>  
---
```

```
mixedBreadcrumbs: BreadcrumbItem[] = [
  { icon: 'home', label: 'Home', url: '/home', active: false },
  { icon: 'folder', label: 'Category', url: '/category', active: false },
  { icon: 'file', label: 'Product', url: '/product', active: true },
];
```

Size Variants

Four different size options (xs, sm, md, lg) for various design contexts.

```
<aava-breadcrumbs [breadcrumbs]="sampleBreadcrumbs" size="xs">
</aava-breadcrumbs>

<aava-breadcrumbs [breadcrumbs]="sampleBreadcrumbs" size="sm">
</aava-breadcrumbs>

<aava-breadcrumbs [breadcrumbs]="sampleBreadcrumbs" size="md">
</aava-breadcrumbs>

<aava-breadcrumbs [breadcrumbs]="sampleBreadcrumbs" size="lg">
</aava-breadcrumbs>
```

```
sampleBreadcrumbs: BreadcrumbItem[] = [
  { icon: 'home', label: 'Home', url: '/home', active: false },
  { icon: 'folder', label: 'Products', url: '/products', active: false },
  {
    icon: 'tag',
    label: 'Category',
    url: '/products/category',
    active: false,
  },
  {
    icon: 'package',
    label: 'Item Details',
    url: '/products/category/item',
  }
];
```

```
active: true,  
},  
];
```

Collapsible Behavior

Smart collapsing for long navigation paths with ellipsis expansion.

```
<aava-breadcrumbs  
[breadcrumbs]="longBreadcrumbs"  
[collapsible]="true"  
[maxVisibleItems]="5"  
>  
</aava-breadcrumbs>  
---  
longBreadcrumbs: BreadcrumbItem[] = [  
{ icon: 'home', label: 'Home', url: '/home', active: false },  
{  
icon: 'building',  
label: 'Organization',  
url: '/organization',  
active: false,  
},  
{ icon: 'folder', label: 'Projects', url: '/projects', active: false },  
{  
icon: 'folder',  
label: 'Web Development',  
url: '/web-dev',  
active: false,  
},  
{ icon: 'folder', label: 'Frontend', url: '/frontend', active: false },
```

```
{ icon: 'folder', label: 'Components', url: '/components', active: false },  
{  
  icon: 'package',  
  label: 'Breadcrumbs',  
  url: '/breadcrumbs',  
  active: true,  
},  
];
```

Custom Separator

Customizable separator icons and sizes.

[Error extracting code]

Dropdown Breadcrumbs

Interactive breadcrumbs with dropdown menus for enhanced navigation options.

[Error extracting code]

Features

Hierarchical Navigation

- Display navigation paths with clear hierarchy
- Support for unlimited breadcrumb levels
- Automatic active state management

Icon Support

- Optional icons for each breadcrumb item
- Consistent icon sizing and coloring
- Icon-text alignment and spacing

Collapsible Behavior

- Smart collapsing for long navigation paths
- Ellipsis expansion on click
- Configurable maximum visible items

- Maintains navigation context

Size Variants

- Four size options: extra small (xs), small (sm), medium (md), and large (lg)
- Consistent typography and spacing across all sizes
- Responsive sizing behavior with appropriate icon scaling

Custom Separators

- Configurable separator icons
- Adjustable separator sizes
- Consistent styling with breadcrumb items

Dropdown Navigation

- Interactive dropdown menus for breadcrumb items
- Support for nested navigation options
- Configurable dropdown styling and positioning
- Icon support within dropdown items
- Click-outside to close functionality

Accessibility

- ARIA attributes for screen readers
- Keyboard navigation support
- Focus management
- Semantic HTML structure

Responsive Design

- Mobile-friendly navigation
- Adaptive collapsing behavior
- Touch-optimized interactions

API Reference

Inputs

[TABLE]

[Property](#) | [Type](#) | [Default](#) | [Description](#)

breadcrumbs | BreadcrumbItem[] | [] | Array of breadcrumb items to display
size | 'sm' | 'md' | 'lg' | 'xs' | 'md' | Size variant for the breadcrumbs
separatorIcon | string | 'chevron-right' | Icon name for breadcrumb separators
separatorSize | number | 14 | Size of separator icons in pixels
dropdownIconSize | number | 14 | Size of dropdown icons in pixels
flyoutStyle | Record<string, string> | undefined | Custom styles for dropdown flyout
customStyles | Record<string, string> | {} | Custom styles for breadcrumb container
collapsible | boolean | true | Enable collapsible behavior for long paths
maxVisibleItems | number | 5 | Maximum items to show before collapsing

[/TABLE]

Interfaces

interface

DropdownItem

{

label

:

string

;

// Dropdown item label

url

:

string

;

// Dropdown item URL

icon

?

:

string

```
;  
// Optional icon for dropdown item  
}  
  
interface  
BreadcrumbItem  
{  
label  
?  
:  
string  
;  
// Optional text label  
icon  
?  
:  
string  
;  
// Optional icon name  
url  
:  
string  
;  
// Required URL for navigation  
active  
:  
boolean  
;  
// Required active state  
dropdownItems
```

```
?  
:  
DropdownItem  
[  
]  
;  
// Optional dropdown items  
dropdownWidth  
?  
:  
number  
;  
// Optional dropdown width  
}
```

Methods

[TABLE]

Method | Parameters | Description

onBreadcrumbClick() | event: Event, index: number | Handle breadcrumb item click and navigation

onEllipsisClick() | None | Expand collapsed breadcrumbs on ellipsis click

onBreadcrumbKeydown() | event: KeyboardEvent, index: number | Handle keyboard navigation

getIconColor() | isLast: boolean | Get appropriate icon color based on state

getOriginalIndex() | displayedIndex: number | Map displayed index to original breadcrumb index

[/TABLE]

Computed Properties

[TABLE]

Property | Type | Description

displayedBreadcrumbs | BreadcrumbItem[] | Current breadcrumbs to display (considering collapse)

shouldCollapse | boolean | Whether breadcrumbs should be collapsed

shouldShowEllipsis | boolean | Whether to show ellipsis for collapsed items

[/TABLE]

CSS Custom Properties

Container Properties

[TABLE]

Property | Description

--breadcrumbs-background | Background color of breadcrumb container

--breadcrumbs-font | Font family and properties for breadcrumb container

[/TABLE]

Item Properties

[TABLE]

Property | Description

--breadcrumbs-item-text | Text color for breadcrumb items

--breadcrumbs-item-border-radius | Border radius for breadcrumb items

--breadcrumbs-item-transition | Transition properties for breadcrumb items

--breadcrumbs-item-hover-text | Text color on hover

--breadcrumbs-item-active-background | Background color for active items

--breadcrumbs-item-disabled-text | Text color for disabled items

--breadcrumbs-item-disabled-cursor | Cursor for disabled items

[/TABLE]

Current Item Properties

[TABLE]

Property | Description

--breadcrumbs-item-current-text | Text color for current/active item

--breadcrumbs-item-current-font-weight | Font weight for current item

[/TABLE]

Size Variants

[TABLE]

Property | Description

--breadcrumbs-size-xs-font | Font properties for extra small size

--breadcrumbs-size-sm-font | Font properties for small size

--breadcrumbs-size-md-font | Font properties for medium size

--breadcrumbs-size-lg-font | Font properties for large size

[/TABLE]

CSS Classes

Container Classes

- .breadcrumb- Main breadcrumb container
- .breadcrumb.xs- Extra small size variant
- .breadcrumb.sm- Small size variant
- .breadcrumb.md- Medium size variant (default)
- .breadcrumb.lg- Large size variant

Item Classes

- .breadcrumb-item- Individual breadcrumb item container
- .breadcrumb-separator- Separator icon styling
- .breadcrumb-ellipsis- Ellipsis container for collapsed items
- .ellipsis- Ellipsis text styling
- .dropdown-container- Container for dropdown breadcrumb items
- .dropdown-button- Button element for dropdown triggers
- .dropdown-menu- Dropdown menu container
- .dropdown-item- Individual dropdown menu items

State Classes

- .active- Active breadcrumb item
- .inactive- Inactive breadcrumb item
- .disabled- Disabled breadcrumb item

Best Practices

Navigation Structure

- Keep breadcrumb paths logical and intuitive
- Use descriptive labels that match page titles
- Ensure URLs are valid and accessible

Icon Usage

- Use consistent icon naming conventions
- Choose icons that clearly represent navigation levels
- Maintain icon-text alignment and spacing

Dropdown Navigation

- Use dropdowns for breadcrumb items that have multiple sub-navigation options
- Provide meaningful labels and icons for dropdown items
- Configure appropriate dropdown width using `dropdownWidth` property
- Use `flyoutStyle` to customize dropdown appearance and positioning

Collapsible Behavior

- Set appropriate `maxVisibleItems` based on design context
- Test collapsing behavior with various path lengths
- Ensure ellipsis expansion provides meaningful navigation

Accessibility

- Provide meaningful labels for screen readers
- Test keyboard navigation thoroughly
- Ensure sufficient color contrast for all states

Responsive Design

- Test breadcrumbs on various screen sizes
- Consider mobile navigation patterns
- Optimize touch targets for mobile devices

Performance

- Avoid excessive breadcrumb levels
- Use efficient navigation logic

- Optimize icon rendering and caching

Use Cases

E-commerce Sites

- Product category navigation
- Search result breadcrumbs
- Shopping cart navigation

Content Management Systems

- Document hierarchy navigation
- Folder structure display
- Content editing paths

Web Applications

- User dashboard navigation
- Settings and configuration paths
- Multi-step form navigation

Documentation Sites

- Documentation hierarchy
- Search result navigation
- Related content linking

Admin Panels

- User management navigation
- System configuration paths
- Report and analytics navigation

Console Applications

- Build tool navigation with dropdown options
- Agent and workflow creation paths
- Quick access to related creation tools

Accessibility

ARIA Support

- Proper aria-current attributes for current page
- Semantic navigation structure
- Screen reader friendly labels

Keyboard Navigation

- Tab navigation through breadcrumb items
- Arrow key navigation between items
- Enter/Space activation for navigation

Focus Management

- Clear focus indicators
- Logical tab order
- Focus restoration after navigation

Screen Reader Support

- Descriptive labels for navigation items
- Context information for collapsed items
- Clear indication of current location

Responsive Behavior

Mobile Optimization

- Touch-friendly tap targets
- Optimized spacing for mobile screens
- Collapsible behavior for limited space

Tablet Adaptation

- Balanced layout for medium screens
- Appropriate icon and text sizing
- Maintained navigation context

Desktop Enhancement

- Full navigation path display
- Hover effects and interactions

- Enhanced visual hierarchy

Browser Support

- Modern Browsers: Full support for all features
- CSS Grid/Flexbox: Required for layout
- ES6+ Features: Required for component functionality
- Angular Router: Required for navigation functionality

PAGINATION

A sophisticated and flexible pagination component designed to handle large datasets with intelligent page number display, multiple visual variants, and comprehensive accessibility features. Perfect for data tables, search results, content lists, and any interface requiring efficient navigation through paginated content. Uses the

ava-pagination-controls

selector for integration.

How to use

```
import
{
  AavaPaginationControlsComponent
}
from
"@aava/play-core"
;
```

Basic Usage

The most basic implementation with default settings and intelligent page number display.

```
<aava-pagination-controls
  [type]="'basic'"
  [currentPage]="'currentPage'"
  [totalPages]="'totalPages'"
  [showNavigationButtons]="'true'"
  (pageChange)="onPageChange($event)"
  [rounded]="'true'"
>
</aava-pagination-controls>
---
currentPage = 1;
totalPages = 20;
```

```
onPageChange(page: number): void {  
    this.currentPage = page;  
}  
}
```

Basic Features

- Smart Ellipsis: Automatically shows ellipsis for large page counts
- Current Page Highlighting: Active page is visually distinguished
- Navigation Buttons: Previous/Next buttons with proper disabled states
- Responsive Design: Adapts to different screen sizes
- Intelligent Truncation: Shows relevant pages around current selection

Size Variants

Five size variants to accommodate different interface densities and visual hierarchy requirements.

```
<aava-pagination-controls  
[type]="'basic'"  
[currentPage]="'basicPage'"  
[totalPages]="'10'"  
[size]="'xl'"  
(pageChange)="onPageChange($event)"  
></aava-pagination-controls>  
  
<aava-pagination-controls  
[type]="'basic'"  
[currentPage]="'basicPage'"  
[totalPages]="'10'"  
[size]="'lg'"  
(pageChange)="onPageChange($event)"  
></aava-pagination-controls>  
  
<aava-pagination-controls  
[type]="'basic'"  
[currentPage]="'basicPage'"
```

```

[totalPages]="10"
[size]="'md'"
(pageChange)="onPageChange($event)"
></aava-pagination-controls>
<aava-pagination-controls
[type]="'basic'"
[currentPage]="basicPage"
[totalPages]="10"
[size]="'sm'"
(pageChange)="onPageChange($event)"
></aava-pagination-controls>
<aava-pagination-controls
[type]="'basic'"
[currentPage]="basicPage"
[totalPages]="10"
[size]="'xs'"
(pageChange)="onPageChange($event)"
></aava-pagination-controls>
---
basicPage = 1;
onPageChange(page: number): void {
  this.basicPage = page;
}

```

Available Sizes

- XSmall- Very compact size for minimal interfaces (12px font)
- Small- Compact size for minimal content and dense interfaces (14px font)
- Medium- Standard size for most pagination scenarios (16px font, default)
- Large- Prominent size for important content (20px font)
- XLarge- Very prominent size for high-visibility interfaces (24px font)

Size Features

- Responsive Typography: Font sizes scale appropriately with size variants
- Consistent Spacing: Button sizes and gaps scale proportionally
- Touch Targets: All sizes maintain minimum 44px touch target requirements
- Visual Hierarchy: Larger sizes provide better emphasis for important pagination

Advanced Features

Icon-Only Navigation

The

iconOnly

property allows navigation buttons to display only icons without text labels, creating a cleaner, more compact design.

Clear Styling

The

clear

property (used internally by unfilled variants) provides transparent button backgrounds for modern, minimalist designs.

Rounded Styling

The

rounded

property applies pill-shaped styling to page number buttons for a softer, more modern appearance.

Page Info Variants

Text-based pagination variants that display current page information instead of page numbers.

<aava-pagination-controls

[type]="pageinfo"

[currentPage]="currentPage"

[totalPages]="totalPages"

(pageChange)="onPageChange(\$event)"

>

</aava-pagination-controls>

```

<aava-pagination-controls
  [type]="'pageinfofilled'"
  [currentPage]="'pageInfoFilledPage'"
  [totalPages]="'10'"
  (pageChange)="onPageInfoFilledChange($event)"
></aava-pagination-controls>

---

currentPage = 1;
totalPages = 10;
pageInfoFilledPage = 1;
onPageChange(page: number): void {
  this.currentPage = page;
}
onPageInfoFilledChange(page: number): void {
  this.pageInfoFilledPage = page;
}

```

Page Info Features

- Text Display: Shows "Page X of Y" format
- Compact Design: Minimal space requirements
- Clear Navigation: Previous/Next buttons for sequential navigation
- Accessibility: Screen reader friendly with descriptive text
- Two Layouts: Full-width and centered variants available

Accessibility

Built-in accessibility features ensuring WCAG compliance and inclusive user experience.

```

<aava-pagination-controls
  [type]="'basic'"
  [currentPage]="'currentPage'"
  [totalPages]="'totalPages'"

```

```

[showNavigationButtons]="true"
(pageChange)="onPageChange($event)"
[rounded]="true"
>
</aava-pagination-controls>
---
currentPage = 1;
totalPages = 20;
onPageChange(page: number): void {
  this.currentPage = page;
}

```

Accessibility Features

- Keyboard Navigation: Full Tab, Arrow key, Enter, and Space support
- ARIA Labels: Proper labels for navigation buttons and page numbers
- Screen Reader Support: Descriptive announcements for page changes
- Focus Management: Clear visual focus indicators
- High Contrast: Enhanced visibility in high contrast mode
- Reduced Motion: Respects user motion preferences

API Reference

Inputs

[TABLE]

Property | Type | Default | Description

currentPage | number | 1 | Current active page number

totalPages | number | 10 | Total number of pages available

type | 'basic' | 'unfilled' | 'basicunfilled' | 'pageinfo' | 'pageinfofilled' | 'basic' | Visual variant of the pagination component

showNavigationButtons | boolean | true | Whether to show Previous/Next navigation buttons

rounded | boolean | false | Whether to apply rounded styling to page number buttons

iconOnly | boolean | false | Whether to show only icons without text labels on navigation buttons

size | 'xs' | 'sm' | 'md' | 'lg' | 'xl' | 'md' | Size variant of the pagination component
customStyles | Record<string, string> | {} | Custom CSS styles to apply to the pagination container
[/TABLE]

Outputs

[TABLE]

Event | Type | Description

pageChange | EventEmitter<number> | Emitted when user navigates to a different page

[/TABLE]

Methods

| Method | Parameters | Return Type | Description |

| ----- | ----- | ----- | ----- |
----- | --- |

||

|

nextPage()

| - |

void

| Navigate to the next page |

|

prevPage()

| - |

void

| Navigate to the previous page |

|

getFontSize()

| - |

string

| Get font size based on current size prop |

|

```
trackByPage(index, page)
|
index: number, page: number \| string
|
string
| TrackBy function for ngFor optimization |
```

Computed Properties

[TABLE]

Property	Type	Description
----------	------	-------------

pages	(number string)[]	Array of page numbers and ellipsis to display
-------	---------------------	---

size	'xs' 'sm' 'md' 'lg' 'xl'	Current size variant of the component
------	----------------------------------	---------------------------------------

[/TABLE]

CSS Custom Properties

| Property | Default | Description |

-----	-----	-----	-----	---
-------	-------	-------	-------	-----

|

--pagination-item-text

Dynamic	Text color for page numbers
---------	-----------------------------

|

--pagination-item-disabled-text

Dynamic	Text color for disabled elements
---------	----------------------------------

||

|

--pagination-container-gap

Dynamic	Gap between pagination elements
---------	---------------------------------

|

--pagination-page-label-color

Dynamic	Color for page label text
---------	---------------------------

```
|  
--pagination-page-label-font-family  
| Dynamic | Font family for page labels |  
|  
--pagination-page-label-font-weight  
| Dynamic | Font weight for page labels |  
|  
--pagination-page-label-line-height  
| Dynamic | Line height for page labels |  
|  
--pagination-page-label-gap  
| Dynamic | Gap between page label elements |
```

Accessibility Guidelines

Keyboard Navigation

- Tab: Navigate to pagination and move between interactive elements
- Arrow Left/Right: Navigate between page numbers (when focused)
- Enter: Activate focused page number or navigation button
- Space: Activate focused page number or navigation button
- Home: Jump to first page (when supported)
- End: Jump to last page (when supported)

Screen Reader Support

- Use descriptive labels that clearly indicate pagination purpose
- Provide context about total number of pages and current position
- Announce page changes and navigation actions
- Include current page information in accessible descriptions
- Use appropriate heading levels for pagination sections

Visual Design

- Maintain sufficient color contrast (4.5:1 minimum) for all states

- Provide clear focus indicators on interactive elements
- Ensure page number buttons meet minimum touch target size (44px)
- Use consistent visual hierarchy across all variants
- Support high contrast and reduced motion preferences

Best Practices

Design Guidelines

- Appropriate Variant Selection: Choose variants based on dataset size and user needs
- Consistent Spacing: Maintain uniform spacing between pagination elements
- Clear Visual Hierarchy: Use proper contrast and sizing for current page indication
- Responsive Design: Ensure pagination works well on all screen sizes
- Loading States: Consider loading indicators during page transitions

Performance

- Efficient Rendering: Use OnPush change detection for optimal performance
- Event Handling: Debounce rapid page changes if triggered programmatically
- Memory Management: Clean up event listeners and subscriptions
- Bundle Optimization: Import only needed variants to minimize bundle size
- Virtual Scrolling: Consider virtual scrolling for very large datasets

User Experience

- Intuitive Navigation: Make it easy to jump to first/last pages
- Page Size Options: Consider adding page size selection for data tables
- Breadcrumb Integration: Use pagination with breadcrumbs for complex navigation
- Search Integration: Combine pagination with search functionality
- URL State: Sync pagination state with URL parameters for bookmarking

Implementation Considerations

- State Management: Properly manage pagination state in your application
- Data Fetching: Implement efficient data loading for each page
- Error Handling: Handle edge cases like invalid page numbers
- Caching: Cache previously visited pages for better performance

- Analytics: Track pagination usage for user behavior insights

Accessibility Implementation

- Semantic HTML: Use proper HTML structure for pagination elements
- ARIA Attributes: Implement appropriate ARIA labels and roles
- Focus Management: Ensure logical tab order and focus indicators
- Screen Reader Testing: Test with actual screen readers
- Keyboard Testing: Verify complete keyboard navigation flow

Technical Notes

Component Architecture

The pagination component uses a modern Angular architecture with:

- PaginationControlsComponent- Main component with multiple variants
- ButtonComponentIntegration for consistent button styling
- OnPush change detectionfor optimal performance
- ViewEncapsulation.Nonefor global CSS variable access

MUI-Style Pagination Logic

The component implements Material-UI style pagination with intelligent page number display:

- Smart Ellipsis: Automatically shows ellipsis for large page counts
- Context-Aware Display: Shows relevant pages around current selection
- Optimal Truncation: Balances information density with usability
- Responsive Behavior: Adapts to different total page counts

Size System

The component provides a comprehensive size system:

- Font Scaling: Each size maps to specific pixel values (12px to 24px)
- Button Scaling: Button dimensions scale proportionally with font size
- Touch Targets: All sizes maintain minimum 44px touch target requirements
- CSS Variables: Uses semantic CSS variables for consistent theming

Variant System

The component supports multiple visual variants:

- Basic Variants: Traditional pagination with filled styling
- Unfilled Variants: Modern design with transparent backgrounds
- Page Info Variants: Text-based alternatives to page numbers
- Icon-Only Options: Clean navigation without text labels

TABS

A comprehensive tab navigation component for organizing content into multiple panels. Features include multiple size variants (xs to xl), icon positioning, bordered styles, container backgrounds, custom styling, scrollable overflow, closeable tabs, and full accessibility support.

How to use

```
import {  
  AavaTabsComponent  
}  
from "@aava/play-core"  
;
```

Basic Usage

The Tabs component allows you to organize content into multiple panels, with only one visible at a time. This is ideal for dashboards, settings, or any interface where users need to switch between related views without navigating away from the page.

- Default behavior: Horizontal layout, first tab active, content panel below.
- Usage: Pass an array of tab objects to the tabs input. Each tab can have a label, content, and optional properties.

```
<aava-tabs  
[tabs]="["  
  { id: 'tab1', label: 'Tab 1', content: 'Content 1' },  
  { id: 'tab2', label: 'Tab 2', content: 'Content 2' },  
  { id: 'tab3', label: 'Tab 3', content: 'Content 3' }  
]"  
activeTabId="tab1"  
></aava-tabs>
```

Variants

Tabs support multiple visual variants for different use cases:

- Default: Standard tabbed navigation with underline indicator and optional icons

- Button: Tabs appear as pill or rounded buttons with optional borders and custom styling
- Icon: Tabs display icons with labels, supporting start/end icon positioning
- Icon Only Square: Compact square tabs showing only icons, ideal for toolbars
- Icon Only Circle: Circular icon-only tabs for minimal interfaces

Choose the variant that best fits your design and use case. The variant

input controls the style, with additional options like

bordered

,

iconPosition

, and

buttonShape

for further customization.

<!-- Default variant with underline -->

<aava-tabs

[tabs]="[

{ id: 'default1', label: 'Home', iconName: 'home', content: 'Welcome to the home page' },

{ id: 'default2', label: 'About', iconName: 'info', content: 'Learn more about us' },

{ id: 'default3', label: 'Contact', iconName: 'phone', content: 'Get in touch' }

]"

variant="default"

activeTabId="default1"

[showContentPanels]="false"

></aava-tabs>

<!-- Button variant - Pill with Border -->

<aava-tabs

[tabs]="[

{ id: 'pill1', label: 'Overview', iconName: 'layout', content: 'Overview content' },

{ id: 'pill2', label: 'Analytics', iconName: 'chart-no-axes-combined', content: 'Analytics content' },

```
{ id: 'pill3', label: 'Reports', iconName: 'file-text', content: 'Reports content' }  
]  
variant="button"  
buttonShape="pill"  
[bordered]="true"  
size="sm"  
activeTabId="pill1"  
[showContentPanels]="false"  
></aava-tabs>  
<!-- Button variant - Rectangle with Border -->  
<aava-tabs  
[tabs]=["  
{ id: 'rect1', label: 'Projects', iconName: 'folder', content: 'Projects content' },  
{ id: 'rect2', label: 'Tasks', iconName: 'check-square', content: 'Tasks content' },  
{ id: 'rect3', label: 'Settings', iconName: 'settings', content: 'Settings content' }  
]  
variant="button"  
buttonShape="rounded"  
[bordered]="true"  
size="sm"  
activeTabId="rect1"  
[showContentPanels]="false"  
></aava-tabs>  
<!-- Icon Only Square variant -->  
<aava-tabs  
[tabs]=["  
{ id: 'icon1', iconName: 'home', label: 'Home', content: 'Home content' },  
{ id: 'icon2', iconName: 'users', label: 'Users', content: 'Users content' },  
{ id: 'icon3', iconName: 'settings', label: 'Settings', content: 'Settings content' }  
"]
```

```
]"
variant="iconOnlySquare"
activeTabId="icon1"
[showContentPanels]="false"
></aava-tabs>
```

Size Variants

Tabs support five size variants to fit different interface densities and use cases:

- Extra Small (xs):Compact tabs for dense interfaces or mobile layouts
- Small (sm):Smaller tabs for secondary navigation or sidebar use
- Medium (md):Default size, balanced for most use cases
- Large (lg):Larger tabs for primary navigation or desktop interfaces
- Extra Large (xl):Maximum size for prominent navigation or accessibility needs

Choose the size that best matches your interface hierarchy and available space. The size

input controls the overall scale of tabs, icons, and spacing.

```
<!-- Extra Large Size -->
<aava-tabs
[tabs]="[

{ id: 'xl1', label: 'Overview', iconName: 'layout', content: 'XL Overview Content' },
{ id: 'xl2', label: 'Analytics', iconName: 'chart-no-axes-combined', content: 'XL Analytics Content' },
{ id: 'xl3', label: 'Reports', iconName: 'file-text', content: 'XL Reports Content' }

]"
variant="button"
buttonShape="rounded"
size="xl"
[bordered]="true"
activeTabId="xl1"
[showContentPanels]="false"
></aava-tabs>
```

```
<!-- Large Size -->

<aava-tabs

[tabs]="[

{ id: 'lg1', label: 'Overview', iconName: 'layout', content: 'LG Overview Content' },

{ id: 'lg2', label: 'Analytics', iconName: 'chart-no-axes-combined', content: 'LG Analytics Content' },

{ id: 'lg3', label: 'Reports', iconName: 'file-text', content: 'LG Reports Content' }

]"

variant="button"

buttonShape="rounded"

size="lg"

[bordered]="true"

activeTabId="lg1"

[showContentPanels]="false"

></aava-tabs>

<!-- Medium Size (Default) -->

<aava-tabs

[tabs]="[

{ id: 'md1', label: 'Overview', iconName: 'layout', content: 'MD Overview Content' },

{ id: 'md2', label: 'Analytics', iconName: 'chart-no-axes-combined', content: 'MD Analytics Content' },

{ id: 'md3', label: 'Reports', iconName: 'file-text', content: 'MD Reports Content' }

]"

variant="button"

buttonShape="rounded"

size="md"

[bordered]="true"

activeTabId="md1"

[showContentPanels]="false"

></aava-tabs>
```

```
<!-- Small Size -->

<aava-tabs

[tabs]="[

{ id: 'sm1', label: 'Overview', iconName: 'layout', content: 'SM Overview Content' },
{ id: 'sm2', label: 'Analytics', iconName: 'chart-no-axes-combined', content: 'SM Analytics Content' },
{ id: 'sm3', label: 'Reports', iconName: 'file-text', content: 'SM Reports Content' }

]"

variant="button"

buttonShape="rounded"

size="sm"

[bordered]="true"

activeTabId="sm1"

[showContentPanels]="false"

></aava-tabs>

<!-- Extra Small Size -->

<aava-tabs

[tabs]="[

{ id: 'xs1', label: 'Overview', iconName: 'layout', content: 'XS Overview Content' },
{ id: 'xs2', label: 'Analytics', iconName: 'chart-no-axes-combined', content: 'XS Analytics Content' },
{ id: 'xs3', label: 'Reports', iconName: 'file-text', content: 'XS Reports Content' }

]"

variant="button"

buttonShape="rounded"

size="xs"

[bordered]="true"

activeTabId="xs1"

[showContentPanels]="false"

></aava-tabs>
```

Tabs with Icons

Add icons to tabs using the

iconName

property in tab objects. Control icon positioning with the

iconPosition

input for flexible layouts.

- Icon Position: Use iconPosition="start" (default) or iconPosition="end" to control icon placement
- Icon-Only Variants: Use iconOnlySquare or iconOnlyCircle variants for compact, icon-only tabs
- When to use: For quick recognition, navigation clarity, or when space is limited

<!-- Icons at start (default) -->

<aava-tabs

```
[tabs]=["  
  { id: 'home1', label: 'Home', iconName: 'home', content: 'Welcome home!' },  
  { id: 'user1', label: 'Profile', iconName: 'user', content: 'Your profile' },  
  { id: 'settings1', label: 'Settings', iconName: 'settings', content: 'App settings' }  
 ]"
```

iconPosition="start"

activeTabId="home1"

[showContentPanels]="false"

></aava-tabs>

<!-- Icons at end -->

<aava-tabs

```
[tabs]=["  
  { id: 'home2', label: 'Home', iconName: 'chevron-right', content: 'Welcome home!' },  
  { id: 'user2', label: 'Profile', iconName: 'chevron-right', content: 'Your profile' },  
  { id: 'settings2', label: 'Settings', iconName: 'chevron-right', content: 'App settings' }  
 ]"
```

iconPosition="end"

activeTabId="home2"

```
[showContentPanels]="false"
></aava-tabs>
<!-- Icon Only Square -->
<aava-tabs
[tabs]="[
{ id: 'home3', label: 'Home', iconName: 'home', content: 'Welcome home!' },
{ id: 'user3', label: 'Profile', iconName: 'user', content: 'Your profile' },
{ id: 'settings3', label: 'Settings', iconName: 'settings', content: 'App settings' }
]"
variant="iconOnlySquare"
activeTabId="home3"
[showContentPanels]="false"
></aava-tabs>
```

Tabs with Badges

Tabs can display badges to indicate counts, notifications, or status. Add a badge

property to any tab object.

- How: Set badge to a number or string.
- When to use: For unread counts, alerts, or status indicators.

[Error extracting code]

Scrollable Tabs

For interfaces with many tabs, enable scrollable

to handle overflow. Scrollable tabs show left/right arrows.

- Scrollable: Use for wide tab sets where horizontal space is available.

[Error extracting code]

Vertical Tabs

Switch to vertical orientation for side navigation or settings panels. Set

orientation="vertical"

- When to use: For sidebar navigation, settings, or when vertical space is preferred.

[Error extracting code]

Custom Styles

Tabs provide multiple styling options for comprehensive customization:

- Tab Row Styling: Use `tabRowWrapperStyles` and `tabRowBackgroundStyles` for custom tab container appearance
- Container Backgrounds: Enable `showTabsContainerBg` for fixed padding and background, or `showContainerBg` for full container styling
- Custom CSS Properties: Use `customStyles` to override design tokens like `--tabs-gap` and color variables
- Active Button Styles: Customize active button appearance with `activeButtonTabStyles` for button variants
- Border Options: Add borders to button and icon variants using the `borderedInput`

These options allow complete visual customization while maintaining accessibility and functionality.

[Error extracting code]

API Reference

Inputs

[TABLE]

Property | Type | Default | Description

`tabs` | `TabItem[]` | `[]` | Array of tab objects (see `TabItem` interface)

`activeTabId` | `string` | `"` | ID of the currently active tab

`variant` | `'default'` | `'button'` | `'icon'` | `'iconOnlySquare'` | `'iconOnlyCircle'` | `'default'` | Visual style of the tabs

`size` | `'xs'` | `'sm'` | `'md'` | `'lg'` | `'xl'` | `'md'` | Size of the tabs

`bordered` | `boolean` | `false` | Add border around tabs (for icon-only variants)

`iconPosition` | `'start'` | `'end'` | `'start'` | Position of icons relative to text

`showContainerBg` | `boolean` | `false` | Show background and border for entire tabs container

`showTabsContainerBg` | `boolean` | `false` | Show tabs container background with fixed padding

`customStyles` | `Record<string, string>` | `{}` | Custom styles for the tabs component

disabled | boolean | false | Disable all tabs
scrollable | boolean | false | Enable horizontal scrolling for overflow
showDropdown | boolean | false | Show dropdown for overflowed tabs
maxVisibleTabs | number | 5 | Max number of visible tabs before overflow
allowTabClose | boolean | false | Allow tabs to be closed
centeredTabs | boolean | false | Center the tabs in the container
fullWidth | boolean | false | Tabs take full width of container
animateTransitions | boolean | true | Animate tab transitions
lazyLoadContent | boolean | false | Lazy load tab content
persistActiveTab | boolean | true | Persist active tab on tab list changes
ariaLabel | string | 'Tabs navigation' | ARIA label for accessibility
activeButtonTabStyles | Record<string, string> | {} | Custom styles for active button tab (button variant)
buttonShape | 'rounded' | 'pill' | 'rounded' | Shape for button variant
showContentPanels | boolean | true | Show content panels below tabs
tabRowWrapperStyles | Record<string, string> | {} | Custom styles for tab row wrapper
tabRowBackgroundStyles | Record<string, string> | {} | Custom styles for tab row background
orientation | 'horizontal' | 'vertical' | 'horizontal' | Orientation of the tabs

[/TABLE]

interface

TabItem

{

id

:

string

;

label

:

string

;

content

?

:

string

;

iconName

?

:

string

;

subtitle

?

:

string

;

disabled

?

:

boolean

;

badge

?

:

string

|

number

;

closeable

?

```
:  
boolean  
;  
}  
}
```

Outputs

[TABLE]

Event | Type | Description

tabChange | EventEmitter<TabItem> | Emitted when the active tab changes

tabClose | EventEmitter<TabItem> | Emitted when a tab is closed

tabsReorder | EventEmitter<TabItem[]> | Emitted when tabs are reordered

dropdownToggle | EventEmitter<boolean> | Emitted when the dropdown is toggled

[/TABLE]

Methods

- `onTabClick(tab: TabItem): void`— Activate a tab programmatically
- `onTabClose(tab: TabItem, event: Event): void`— Close a tab programmatically
- `toggleDropdown(): void`— Toggle the dropdown menu
- `scrollLeft(): void`/`scrollRight(): void`— Scroll the tab list

Accessibility

The Tabs component follows WAI-ARIA accessibility guidelines and provides comprehensive keyboard navigation:

- Keyboard Navigation: Full support for Tab, Arrow keys (Left/Right/Up/Down), Enter, and Space
- ARIA Roles: Propertablist,tab, andtabpanelroles for screen reader compatibility
- ARIA Attributes:aria-selected,aria-controls,aria-labelledby, andaria-disabledfor state communication
- Focus Management: Visible focus indicators and proper tab sequence navigation
- Screen Reader Support: Descriptive labels, state announcements, and semantic HTML structure
- Keyboard Shortcuts: Arrow keys for tab navigation, Enter/Space for activation
- High Contrast: Enhanced borders and indicators for visibility in high contrast mode

Design Tokens & Theming

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

Available Design Tokens for Tabs

[TABLE]

Token | Purpose | Default Value

--tabs-gap | Spacing between tabs | Theme-based

[/TABLE]

[TABLE]

Token | Purpose | Default Value

--tabs-tab-color | Color for inactive tab text | Theme-based

--tabs-tab-active-color | Color for active tab text | Theme-based

--tabs-tab-disabled-color | Color for disabled tab text | Theme-based

[/TABLE]

[TABLE]

Token | Purpose | Default Value

--tabs-icon-size | Default size for tab icons | Theme-based

[/TABLE]

Best Practices

Design Guidelines

- Semantic Organization: Group related content logically and use clear, descriptive tab labels
- Tab Count: Limit to 5-7 tabs for optimal usability; use scrollable or dropdown for more
- Variant Selection: Choose variants based on context (default for content, button for actions, icon-only for compact)
- Size Selection: Use appropriate sizes (xs-xl) based on interface hierarchy and available space
- Icon Positioning: Consider iconPosition="end" for navigation arrows or action indicators
- Border Usage: Apply bordered input for better visual separation in dense interfaces
- Container Backgrounds: Use showTabsContainerBg or showContainerBg for visual grouping
- Visual Hierarchy: Use active states, badges, and icons to guide user attention appropriately
- Responsive Design: Consider vertical orientation or dropdown behavior for mobile devices

- Consistent Sizing: Match tab sizes to surrounding interface elements and content density

Accessibility

- Meaningful Labels: Provide clear, descriptive labels that explain each tab's content or purpose
- Unique IDs: Ensure each tab has a unique ID for proper ARIA relationships
- Keyboard Support: Test complete keyboard navigation flow including arrow keys and activation
- Screen Reader Testing: Verify proper announcement of tab states, content, and navigation
- Focus Indicators: Maintain clear visual focus states for keyboard navigation
- Color Contrast: Ensure sufficient contrast for all tab states and variants
- Reduced Motion: Respect user preferences for reduced motion in transitions

Performance

- Lazy Loading: Enable `lazyLoadContent` for tabs with heavy content to improve initial load
- Content Optimization: Only render active tab content unless pre-loading is necessary
- Icon Efficiency: Use icon systems efficiently and avoid loading unnecessary icon variants
- Smooth Scrolling: Use hardware-accelerated scrolling for large tab sets
- Event Handling: Debounce rapid tab switching and optimize event listeners
- Bundle Optimization: Import only needed variants and features to minimize bundle size
- Memory Management: Properly clean up event listeners and subscriptions in closeable tabs

MENU

A flexible and feature-rich dropdown menu component for creating context menus, navigation dropdowns, and action menus. Supports icons, descriptions, multi-column layout, various positioning options, and full accessibility.

How to use

```
import
{
  AavaMenuComponent
}
from
"@aava/play-core"
;
```

Basic Usage

The Menu component provides a dropdown interface that can be positioned relative to a trigger element. It supports various menu items with different states and configurations.

- Default behavior: Bottom positioning, single column layout, with icons and titles.
- Usage: Pass an array of menu items to the `items` input and control visibility with the `visible` input.

```
<div class="menu-container">
  <button (click)="toggleMenu()">Open Menu</button>
  <aava-menu
    [items]="menuItems"
    [visible]="showMenu"
    (itemSelected)="handleItemSelected($event)">
  </aava-menu>
</div>
---
import { Component } from "@angular/core";
import { AavaMenuComponent } from "@aava/play-core";
@Component({
```

```

selector: "app-menu-basic",
standalone: true,
imports: [AavaMenuComponent],
})

export class MenuBasicComponent {
showMenu = false;
menuItems = [
{ label: "Dashboard", route: "/dashboard" },
{ label: "Profile", route: "/profile" },
{ label: "Settings", route: "/settings" },
{ label: "Help", route: "/help" },
{ label: "Logout", route: "/logout" },
];
toggleMenu() {
this.showMenu = !this.showMenu;
}
handleItemSelected(event: any) {
console.log("Selected:", event.label);
this.showMenu = false;
}
}

```

With Icons

Menu items can display icons alongside their labels for better visual recognition and user experience.

- How: Add an icon property to menu items with the icon name.
- When to use: For quick recognition, consistent visual language, or when space allows.

```

<div class="menu-container">

<button (click)="toggleMenu()">Menu with Icons</button>

<aava-menu
[items]="menuItems"

```

```

[visible]="showMenu"

[displayOptions]="{ showIcon: true, showTitle: true }"

(itemSelected)="handleItemSelected($event)">

</aava-menu>

</div>

---

import { Component } from "@angular/core";
import { AavaMenuComponent } from "@aava/play-core";
@Component({
  selector: "app-menu-with-icons",
  standalone: true,
  imports: [AavaMenuComponent],
})
export class MenuWithIconsComponent {
  showMenu = false;
  menuItems = [
    { label: "Dashboard", icon: "home", route: "/dashboard" },
    { label: "Profile", icon: "user", route: "/profile" },
    { label: "Settings", icon: "settings", route: "/settings" },
    { label: "Notifications", icon: "bell", route: "/notifications" },
    { label: "Help", icon: "help-circle", route: "/help" },
    { label: "Logout", icon: "log-out", route: "/logout" },
  ];
  toggleMenu() {
    this.showMenu = !this.showMenu;
  }
  handleItemSelected(event: any) {
    console.log("Selected:", event.label);
  }
}

```

```
this.showMenu = false;  
}  
}
```

With Descriptions

Menu items can include descriptions to provide additional context or information about each option.

- How: Add a `description` property to menu items.
- When to use: For complex menus where items need additional explanation or context.

```
<div class="menu-container">  
  
<button (click)="toggleMenu()">Menu with Descriptions</button>  
  
<aava-menu  
  
[items]="menuItems"  
  
[visible]="showMenu"  
  
[displayOptions]="{ showIcon: true, showTitle: true, showDescription: true }"  
  
(itemSelected)="handleItemSelected($event)">  
  
</aava-menu>  
  
</div>  
  
---  
  
import { Component } from "@angular/core";  
  
import { AavaMenuComponent } from "@aava/play-core";  
  
@Component({  
  
  selector: "app-menu-with-descriptions",  
  
  standalone: true,  
  
  imports: [AavaMenuComponent],  
  
})  
  
export class MenuWithDescriptionsComponent {  
  
  showMenu = false;  
  
  menuItems = [  
  
    {  
  
      label: "Dashboard",  
  
      description: "The main dashboard for monitoring system health and performance."  
    },  
  
    {  
  
      label: "Logs",  
  
      description: "A detailed log viewer showing application logs and error logs."  
    },  
  
    {  
  
      label: "Metrics",  
  
      description: "Real-time metrics and performance indicators for the system."  
    },  
  
    {  
  
      label: "Configurations",  
  
      description: "A view of current system configurations and settings."  
    },  
  
    {  
  
      label: "Database",  
  
      description: "A database management interface for querying and managing data."  
    },  
  
    {  
  
      label: "APIs",  
  
      description: "A list of available APIs and their documentation."  
    },  
  
    {  
  
      label: "Cloud Services",  
  
      description: "A view of connected cloud services and their status."  
    },  
  
    {  
  
      label: "Logs",  
  
      description: "A detailed log viewer showing application logs and error logs."  
    },  
  
    {  
  
      label: "Metrics",  
  
      description: "Real-time metrics and performance indicators for the system."  
    },  
  
    {  
  
      label: "Configurations",  
  
      description: "A view of current system configurations and settings."  
    },  
  
    {  
  
      label: "Database",  
  
      description: "A database management interface for querying and managing data."  
    },  
  
    {  
  
      label: "APIs",  
  
      description: "A list of available APIs and their documentation."  
    },  
  
    {  
  
      label: "Cloud Services",  
  
      description: "A view of connected cloud services and their status."  
    }  
  ]  
}  
};
```

```
icon: "home",
description: "View your main dashboard and overview",
route: "/dashboard",
},
{
label: "Profile",
icon: "user",
description: "Manage your account settings and preferences",
route: "/profile",
},
{
label: "Settings",
icon: "settings",
description: "Configure application settings and options",
route: "/settings",
},
{
label: "Help",
icon: "help-circle",
description: "Get help and support documentation",
route: "/help",
},
];
toggleMenu() {
this.showMenu = !this.showMenu;
}
handleItemSelected(event: any) {
console.log("Selected:", event.label);
```

```
this.showMenu = false;  
}  
}
```

Disabled Items

Menu items can be disabled to indicate unavailable options or actions that require certain conditions.

- How: Set `disabled`: true on menu items.
- When to use: For conditional actions, permission-based menus, or unavailable features.

```
<div class="menu-container">  
  
<button (click)="toggleMenu()">Menu with Disabled Items</button>  
  
<aava-menu  
  
[items]="menuItems"  
  
[visible]="showMenu"  
  
[displayOptions]="{ showIcon: true, showTitle: true, showDescription: true }"  
  
(itemSelected)="handleItemSelected($event)">  
  
</aava-menu>  
  
</div>  
  
---  
  
import { Component } from "@angular/core";  
  
import { AavaMenuComponent } from "@aava/play-core";  
  
@Component({  
  
selector: "app-menu-disabled-items",  
  
standalone: true,  
  
imports: [AavaMenuComponent],  
  
})  
  
export class MenuDisabledItemsComponent {  
  
showMenu = false;  
  
menuItems = [  
  
{  
  
label: "Dashboard",
```

```
icon: "home",
description: "View your main dashboard",
route: "/dashboard",
},
{
label: "Profile",
icon: "user",
description: "Manage your account settings",
route: "/profile",
},
{
label: "Premium Features",
icon: "star",
description: "Upgrade to access premium features",
disabled: true,
},
{
label: "Advanced Analytics",
icon: "bar-chart",
description: "Advanced analytics (requires premium)",
disabled: true,
},
{
label: "Help",
icon: "help-circle",
description: "Get help and support",
route: "/help",
},
```

```
];
toggleMenu() {
  this.showMenu = !this.showMenu;
}
handleItemSelected(event: any) {
  console.log("Selected:", event.label);
  this.showMenu = false;
}
}
```

Features

Flexible Positioning

- 8 Position Options: top, bottom, left, right, and their start/end variants
- Alignment Control: start, center, end alignment within positioned area
- Offset Configuration: Customizable spacing from trigger element
- Auto Flip: Automatic position adjustment when space is limited

Rich Content Support

- Icons: Lucide icons with customizable size and colors
- Descriptions: Additional text for context and explanation
- Active States: Visual indication of currently active/selected items
- Disabled States: Clear indication of unavailable options

Layout Options

- Multi-Column: Configurable column layout for extensive menus
- Responsive Design: Adapts to different screen sizes and orientations
- Flexible Sizing: Automatic width adjustment based on content
- Custom Styling: Extensive CSS custom properties for theming

Accessibility

- Keyboard Navigation: Full keyboard support with arrow keys
- Screen Reader: Proper ARIA attributes and semantic structure

- Focus Management: Clear focus indicators and logical tab order
- High Contrast: Support for high contrast mode and color preferences

API Reference

Inputs

[TABLE]

Property	Type	Description
items	MenuItem[] []	Array of menu items to display
visible	boolean	false Controls menu visibility
itemsPerColumn	number	3 Number of items per column in multi-column layout
positionConfig	MenuPositionConfig	See default config below Configuration for menu positioning
displayOptions	MenuItemDisplayOptions	See default options below Configuration for item display
customStyles	Record<string, string> {}	Custom CSS styles to apply to the menu

[/TABLE]

Outputs

[TABLE]

Property	Type	Description
itemSelected	EventEmitter<{route?: string; label: string; item: MenuItem}>	Emitted when a menu item is clicked

MenuItem Interface

interface

MenuItem

{

label

:

string

;

// Display text for the menu item

```
description
?
:
string
;
// Optional description text

route
?
:
string
;
// Optional route for navigation

icon
?
:
string
;
// Optional icon name (Lucide)

disabled
?
:
boolean
;
// Whether the item is disabled

divider
?
:
boolean
;
```

```
// Whether to show a divider line
customData
?
:
Record
<
string
,
unknown
>
;

// Custom data for the item
}
```

MenuPositionConfig Interface

```
interface
MenuPositionConfig
{
position
?
:
MenuPosition
;
// Position relative to trigger
alignment
?
:
MenuAlignment
;
// Alignment within positioned area
```

```
offset
?
:
number
;
// Distance from trigger element
autoFlip
?
:
boolean
;
// Auto-adjust position if needed
}

type
MenuPosition
=
|
"top"
|
"bottom"
|
"left"
|
"right"
|
"top-start"
|
"top-end"
|
```

```
"bottom-start"
```

```
|
```

```
"bottom-end"
```

```
;
```

```
type
```

```
MenuAlignment
```

```
=
```

```
"start"
```

```
|
```

```
"center"
```

```
|
```

```
"end"
```

```
;
```

MenuItemDisplayOptions Interface

```
interface
```

```
MenuItemDisplayOptions
```

```
{
```

```
showIcon
```

```
?
```

```
:
```

```
boolean
```

```
;
```

```
// Whether to show icons
```

```
showTitle
```

```
?
```

```
:
```

```
boolean
```

```
;
```

```
// Whether to show item titles
```

```
showDescription
?
:
boolean
;
// Whether to show descriptions

iconSize
?
:
number
;
// Size of icons in pixels

titleWeight
?
:
"normal"
|
"medium"
|
"semibold"
|
"bold"
;

// Font weight for titles

descriptionSize
?
:
"xs"
|
```

```
"sm"  
|  
"md"  
|  
"lg"  
;  
// Font size for descriptions  
}
```

Default Configurations

```
// Default position config
```

```
{  
position  
:  
'bottom'  
,
```

```
alignment
```

```
:  
'start'
```

```
,  
offset  
:  
8  
,
```

```
}
```

```
// Default display options
```

```
{  
showIcon  
:  
true
```

```
,  
showTitle  
:  
true  
,  
showDescription  
:  
true  
,  
iconSize  
:  
24  
,  
titleWeight  
:  
'medium'  
,  
descriptionSize  
:  
'sm'  
,
```

CSS Custom Properties

The component uses CSS custom properties for theming:

[TABLE]

Property | Description

--menu-min-width | Minimum width of the menu

--menu-background | Background color of the menu

--menu-border-radius | Border radius of the menu

--menu-shadow | Box shadow of the menu
--menu-border | Border of the menu
--menu-padding | Padding inside the menu
--menu-margin-top | Top margin for positioning
--menu-transform-y | Transform distance for animations
--menu-transition | Transition timing for animations
--menu-z-index | Z-index for layering
--menu-columns-gap | Gap between columns
--menu-column-min-width | Minimum width of each column
--menu-item-gap | Gap between icon and content
--menu-item-padding | Padding of menu items
--menu-item-border-radius | Border radius of menu items
--menu-item-background | Background of menu items
--menu-item-color | Text color of menu items
--menu-item-hover-background | Background on hover
--menu-item-hover-color | Text color on hover
--menu-item-active-background | Background when active
--menu-item-active-color | Text color when active
--menu-item-disabled-background | Background when disabled
--menu-item-disabled-color | Text color when disabled
--menu-item-margin | Margin of menu items
--menu-item-first-margin-top | Top margin of first item
--menu-item-last-margin-bottom | Bottom margin of last item
--menu-item-transition | Transition timing for items
--menu-item-icon-size | Size of item icons
--menu-item-icon-color | Color of item icons
--menu-item-icon-active-color | Color of active item icons
--menu-item-icon-transition | Transition timing for icons
--menu-item-hover-icon-filter | Icon filter on hover

--menu-item-active-icon-filter | Icon filter when active
--menu-item-content-gap | Gap between title and description
--menu-item-title-font-weight | Font weight of titles
--menu-item-title-font-size | Font size of titles
--menu-item-title-color | Color of titles
--menu-item-title-active-color | Color of active titles
--menu-item-title-active-font-weight | Font weight of active titles
--menu-item-title-line-height | Line height of titles
--menu-item-title-transition | Transition timing for titles
--menu-item-description-font-size | Font size of descriptions
--menu-item-description-color | Color of descriptions
--menu-item-description-active-color | Color of active descriptions
--menu-item-description-line-height | Line height of descriptions
--menu-item-description-transition | Transition timing for descriptions

[/TABLE]

Best Practices

Content Guidelines

- Clear Labels: Use concise, descriptive labels for menu items
- Consistent Icons: Use consistent iconography throughout the menu
- Helpful Descriptions: Provide context when items need explanation
- Logical Grouping: Group related items together in the menu

Interaction Design

- Trigger Elements: Ensure trigger elements are clearly identifiable
- Positioning: Choose positioning that doesn't obscure important content
- Keyboard Support: Test keyboard navigation thoroughly
- Touch Support: Consider touch interactions on mobile devices

Accessibility

- ARIA Labels: Provide appropriate ARIA labels for screen readers

- Focus Management: Ensure proper focus handling when menu opens/closes
- Color Contrast: Maintain sufficient contrast for all text and icons
- Motion Sensitivity: Respect user motion preferences

Performance

- Efficient Rendering: Use OnPush change detection for better performance
- Event Handling: Properly handle click events to prevent bubbling
- Memory Management: Clean up event listeners and subscriptions
- Lazy Loading: Consider lazy loading for large menus

Use Cases

Context Menus

Right-click context menus for actions on selected items:

```
<  
aava  
-  
menu  
[  
items  
]  
=  
"contextMenuItems"  
[  
visible  
]  
=  
"showContextMenu"  
[  
positionConfig  
]
```

```
=  
  "{ position: 'bottom-start' }"  
(  
itemSelected  
)  
=  
"handleContextMenuAction($event)"  
>  
<  
/  
aava  
  
-  
menu  
>
```

Navigation Dropdowns

Dropdown navigation menus for main navigation:

```
<  
aava  
  
-  
menu  
[  
items  
]  
=  
"navigationItems"  
[  
visible  
]  
=
```

```
"showNavMenu"
[
  positionConfig
]
=
"{ position: 'bottom', alignment: 'center' }"
[
  displayOptions
]
=
"{ showIcon: true, showDescription: false }"
>
<
/
aava
-
menu
>
```

Action Menus

Action menus for buttons or toolbar items:

```
<
aava
-
menu
[
  items
]
=
"actionItems"
```

```
[  
  visible  
]  
=  
"showActionMenu"  
[  
  positionConfig  
]  
=  
"{ position: 'bottom-end' }"  
[  
  itemsPerColumn  
]  
=  
"1"  
>  
<  
/  
aava  
-  
menu  
>
```

Application Launchers

Multi-column menus for application or feature launchers:

```
<  
aava  
-  
menu  
[
```

```
items
]
=
"appItems"
[
visible
]
=
"showAppMenu"
[
positionConfig
]
=
"{ position: 'bottom', alignment: 'center' }"
[
itemsPerColumn
]
=
"4"
[
displayOptions
]
=
"{ showIcon: true, showDescription: true }"
>
<
/
aava
-
```

menu

>

Accessibility Guidelines

Keyboard Navigation

The menu supports full keyboard navigation:

- Enter/Space: Activate the currently focused menu item
- Arrow Keys: Navigate between menu items
- Escape: Close the menu
- Tab: Move focus to next focusable element

Screen Reader Support

The component provides comprehensive screen reader support:

- ARIA Labels: Proper labeling for menu and menu items
- Role Attributes: Correct ARIA roles for menu structure
- State Announcements: Clear announcements of menu state changes
- Focus Indicators: Visible focus indicators for keyboard users

Color and Contrast

- WCAG Compliance: All text and icons meet WCAG AA contrast ratios
- High Contrast Mode: Component works with system high contrast settings
- Color Independence: Information is not conveyed by color alone

Motion and Animation

- Respects Preferences: Animation respects user motion preferences
- Reduced Motion: Provides alternative interaction for motion-sensitive users
- Clear Feedback: Visual feedback is immediate and clear

LINK

A simple and effective action link component for navigation and interactive actions. Features semantic color variants, multiple size options, optional underline styling, and support for custom hex colors with smooth hover transitions.

How to use

```
import {  
  AavaLinkComponent  
}  
from "@aava/play-core"  
;
```

Basic Usage

Simple link implementations with customizable labels, icons, and styling.

```
<!-- Basic Usage -->  
  
<aava-link label="Get Started" color="primary"></aava-link>  
  
<!-- With Icon -->  
  
<aava-link  
  label="Link with Icon"  
  [addIcon]="true"  
  arrowDirection="right"  
  color="primary">  
  </aava-link>  
  
<!-- Custom Styles -->  
  
<aava-link  
  label="Custom Styled Link"  
  [customStyles]="{'font-weight': 'bold'}"  
  color="info">  
  </aava-link>
```

Colors

Semantic color variants and custom hex color support for different contexts.

```
<!-- Primary Color -->  
  
<aava-link label="Primary Link" color="primary"></aava-link>  
  
<!-- Success Color -->  
  
<aava-link label="Success Link" color="success"></aava-link>  
  
<!-- Warning Color -->  
  
<aava-link label="Warning Link" color="warning"></aava-link>  
  
<!-- Danger Color -->  
  
<aava-link label="Danger Link" color="danger"></aava-link>  
  
<!-- Info Color -->  
  
<aava-link label="Info Link" color="info"></aava-link>  
  
<!-- Custom Hex Color -->  
  
<aava-link label="Custom Link" color="#7C3AED"></aava-link>
```

Available Color Variants

- Default: Standard link color
- Primary: Brand primary color
- Success: Green for success actions
- Warning: Orange for caution actions
- Danger: Red for destructive actions
- Info: Blue for informational links
- Custom Hex: Any valid hex color value (e.g., #7C3AED, #14B8A6)

Sizes

Multiple size options for different layout contexts and visual hierarchy.

```
<!-- Small Size -->  
  
<aava-link label="Small Link" size="sm" color="primary"></aava-link>  
  
<!-- Medium Size (Default) -->  
  
<aava-link label="Medium Link" size="md" color="primary"></aava-link>
```

```
<!-- Large Size -->  
<aava-link label="Large Link" size="lg" color="primary"></aava-link>
```

Size Options

- Small: Compact links for dense layouts
- Medium: Standard size for most use cases (default)
- Large: Prominent links for primary actions

Underline

Optional underline styling for enhanced visual emphasis and accessibility.

```
<!-- Without Underline (Default) -->  
<aava-link label="No Underline" [underline]="false" color="primary"></aava-link>  
<!-- With Underline -->  
<aava-link label="With Underline" [underline]="true" color="primary"></aava-link>
```

Features

Color System

- Semantic color variants (primary, success, warning, danger, info)
- Custom hex color support with automatic validation
- Consistent hover effects across all color variants

Icon Support

- Optional chevron icons with directional control
- Left or right arrow positioning
- Size-responsive icon scaling

Customization

- Custom CSS styles through `customStyles` input
- Flexible sizing options (sm, md, lg)
- Optional underline styling for accessibility

Accessibility

- Proper ARIA attributes and role support

- Keyboard navigation (Enter/Space key support)
- Screen reader friendly labels
- Sufficient color contrast for all variants

API Reference

Inputs

[TABLE]

Property | Type | Default | Description

label | string | 'Action Link' | The text content of the link

color | 'success' | 'warning' | 'danger' | 'info' | 'default' | 'primary' | string | 'default' | Color variant or custom hex color

size | 'sm' | 'md' | 'lg' | 'md' | Size variant for the link

underline | boolean | false | Whether to show underline styling

href | string | '' | URL for navigation

addIcon | boolean | false | Whether to show an icon with the link

arrowDirection | 'right' | 'left' | 'left' | Direction of the arrow icon

customStyles | Record<string, string> | {} | Custom CSS styles to apply

[/TABLE]

Outputs

[TABLE]

Event | Type | Description

userClick | EventEmitter<Event> | Emitted when the link is clicked

[/TABLE]

Methods

[TABLE]

Method | Parameters | Return Type | Description

isHexColor() | color: string | boolean | Validates if a color string is a valid hex color

getLinkStyles() | None | Record<string, string> | Returns computed styles including custom hex colors

anchorClick() | event: Event | void | Handles click events and emits userClick event

[/TABLE]

Properties

[TABLE]

Property | Type | Description

separatorIcon | string | Icon name for the arrow (default: 'chevron-right')

separatorSize | number | Size of the arrow icon (responsive to link size)

safeHref | SafeUrl | Sanitized URL for secure navigation

[/TABLE]

CSS Custom Properties

[TABLE]

Property | Default | Description

--link-size-sm-font | Theme-based | Font size for small links

--link-size-md-font | Theme-based | Font size for medium links

--link-size-lg-font | Theme-based | Font size for large links

--link-primary-text | Theme-based | Primary color variant

--link-danger-text | Theme-based | Danger color variant

--link-success-text | Theme-based | Success color variant

--link-warning-text | Theme-based | Warning color variant

--link-info-text | Theme-based | Info color variant

--link-active-text-decoration | Theme-based | Underline decoration style

[/TABLE]

Best Practices

Design Guidelines

- Use semantic colors to convey meaning and context
- Reserve large size (lg) for primary actions and important navigation
- Consider underline for accessibility and emphasis
- Maintain consistent sizing within related link groups
- Use custom hex colors sparingly to maintain design consistency

- Use icons strategically to enhance navigation clarity
- Choose appropriate arrow direction based on content flow

Accessibility

- Ensure sufficient color contrast for all color variants
- Use descriptive label text that clearly indicates the action or destination
- Consider underline styling for better visual accessibility
- Test with screen readers to ensure proper announcement
- Provide adequate spacing between links for touch interactions

Technical Notes

Color System

The component supports both predefined semantic colors and custom hex values:

- Semantic colors use CSS custom properties for theme consistency
- Custom hex colors are validated using `isHexColor()` method
- Custom colors are applied directly via inline styles
- Hover effects maintain consistency across all color variants

Icon System

- Icons are conditionally rendered based on `addIcon` property
- Arrow direction is controlled by `arrowDirectionInput`
- Icon sizes automatically scale with link size variants
- Icons are excluded when underline styling is enabled

Styling Architecture

- Uses CSS classes for size and variant styling
- Combines semantic classes with conditional custom styles
- Smooth transitions for interactive states
- Maintains proper text decoration control

ViewEncapsulation

The component uses

`ViewEncapsulation.None`

to allow global styling:

- Enables theme-wide link styling consistency
- Allows CSS custom property inheritance
- Supports integration with design system tokens

STEPPER-INPUT

A sophisticated Angular multi-step navigation component for creating step-by-step wizards, form flows, and progress workflows. Features animated progress indicators, interactive navigation, and flexible layout options for guiding users through complex processes.

How to use

```
import {  
  AvaStepperComponent  
}  
from "@aava/play-core"  
;
```

Basic Usage

The most basic implementation with step labels and default horizontal orientation.

```
<aava-button  
  label="Show Success Toast"  
  variant="success"  
  (userClick)="showSuccessToast()"></aava-button>  
  
---  
// Inject the toast service in your component  
constructor(private toastService: ToastService) {}  
  
// Show different types of toasts  
showSuccessToast() {  
  this.toastService.success({  
    title: 'Successfully created!',  
    message: 'Your changes have been saved successfully',  
    duration: 2000,  
    customWidth: '400px',
```

```
    design: 'modern',  
    size: 'large',  
  });  
}
```

Sizes

Three size options to accommodate different interface densities and visual hierarchies.

```
<aava-button  
  label="Show Success Toast"  
  variant="success"  
  (userClick)="showSuccessToast()"></aava-button>  
  
<aava-button  
  label="Show Error Toast"  
  variant="danger"  
  (userClick)="showErrorToast()"></aava-button>  
  
<aava-button  
  label="Show Warning Toast"  
  variant="warning"  
  (userClick)="showWarningToast()"></aava-button>  
  
<aava-button  
  label="Show Info Toast"  
  variant="info"  
  (userClick)="showInfoToast()"></aava-button>  
  
<aava-button  
  label="Show Default Toast"  
  variant="secondary">
```

```
(userClick)="showDefaultToast()"

></aava-button>

---

// Inject the toast service in your component

constructor(private toastService: ToastService) {}

// Show different toast variants

showSuccessToast() {

  this.toastService.success({
    title: 'Successfully created!',
    message: 'Your changes have been saved successfully',
    duration: 2000,
    customWidth: '300px',
    design: 'modern',
    size: 'small',
  });
}

showErrorToast() {

  this.toastService.error({
    title: 'Error Occurred',
    message: 'Connection error. Unable to connect to the server at present',
    duration: 2000,
    customWidth: '300px',
    design: 'modern',
    size: 'small',
  });
}

showWarningToast() {

  this.toastService.warning({

```

```
title: 'Warning Occurred',
message: 'Please review your input carefully before proceeding.',
duration: 2000,
customWidth: '300px',
design: 'modern',
size: 'small',
});
}

showInfoToast() {
this.toastService.info({
title: 'Action Required',
message: 'Please review your input carefully before proceeding.',
duration: 2000,
customWidth: '300px',
design: 'modern',
size: 'small',
});
}

showDefaultToast() {
this.toastService.default({
title: 'Default Toast Occurred',
message: 'This is a default toast with neutral styling.',
duration: 2000,
customWidth: '300px',
design: 'modern',
size: 'small',
});
}

showCustomToast() {
```

```
this.toastService.custom({  
  title: 'Custom Toast',  
  message: 'This is a fully customizable toast with unique styling.',  
  customBackground: 'var(--color-primary)',  
  customTextColor: 'var(--color-text-primary)',  
  customWidth: '300px',  
  design: 'modern',  
  size: 'small',  
});  
}
```

Available Sizes

- xs: Extra small for minimal interfaces
- sm: Small for compact layouts
- md: Medium size for most use cases (default)
- lg: Large for prominent workflows and accessibility

Icon Variants

Enhanced stepper with custom icons for each step, providing better visual context and user guidance.

```
<aava-button  
  label="Show Small Success"  
  variant="success"  
  size="small"  
  (userClick)="showSmallSuccess()"></aava-button>  
  
<aava-button  
  label="Show Medium Warning"  
  variant="warning"  
  size="small"  
  (userClick)="showMediumWarning()">
```

```
></aava-button>

<aava-button
    label="Show Large Success"
    variant="danger"
    size="small"
    (userClick)="showLargeError()"

></aava-button>

---

// Inject the toast service in your component
constructor(private toastService: ToastService) {}

// Show different toast sizes

showSmallSuccess() {
    this.toastService.success({
        title: 'Successfully created!',
        message: 'Your changes have been saved successfully',
        duration: 1000,
        customWidth: '300px',
        design: 'modern',
        size: 'small',
    });
}

showMediumWarning() {
    this.toastService.warning({
        title: 'Action Required',
        message: 'Incomplete fields. Please fill in all required information.',
        duration: 1000,
        customWidth: '350px',
        design: 'modern',
        size: 'medium',
    });
}
```

```
});

}

showLargeError() {
  this.toastService.error({
    title: 'Error Occurred',
    message: 'Connection error. Unable to connect to the server at present',
    duration: 1000,
    customWidth: '400px',
    design: 'modern',
    size: 'large',
  });
}
```

Icon Features

- Custom Step Icons: Replace numeric indicators with meaningful Lucide icons
- Contextual Guidance: Icons provide immediate visual context for each step
- State Awareness: Icons adapt to active, completed, and disabled states
- Consistent Sizing: Icons scale appropriately with stepper size variants
- Color Theming: Icons inherit stepper theme colors automatically
- Accessibility: Proper ARIA labels and descriptions for screen readers

Orientation

Flexible layout options for different design requirements and content arrangements.

Horizontal Orientation

- Default layout: Steps arranged left to right
- Space efficient: Ideal for wide containers
- Progress visualization: Horizontal progress lines
- Label positioning: Labels positioned below step circles

Vertical Orientation

- Alternative layout: Steps arranged top to bottom

- Sidebar friendly: Perfect for narrow containers and sidebars
- Vertical progress: Connecting lines flow downward
- Inline labels: Labels positioned next to step circles

Interactive Navigation

Control user interaction and step accessibility with interactive navigation options.

Interactive Features

- Click navigation: Jump to any accessible step by clicking
- Keyboard navigation: Enter and Space key support
- Disabled steps: Prevent navigation to specific steps
- Step validation: Control progression based on form validity
- Non-interactive mode: Display-only stepper for progress indication

Event Handling

Comprehensive event system for tracking step changes and workflow completion.

Available Events

- stepChange: Emitted when user navigates to a different step
- stepperComplete: Emitted when the final step is reached
- Step validation: Handle step transitions with custom validation logic

Accessibility

Built-in accessibility features ensuring WCAG compliance and inclusive user experience.

Accessibility Features

- Keyboard navigation: Tab, Enter, and Space key support
- ARIA labels: Descriptive labels for screen readers
- Role attributes: Proper button roles for interactive elements
- Focus management: Clear visual focus indicators
- Step announcements: Screen reader notifications for step changes
- High contrast: Support for high contrast mode preferences

API Reference

Inputs

[TABLE]

Property | Type | Default | Description

steps | (string | StepperStep)[] | [] | Array of step labels or step objects to display

currentStep | number | 0 | Index of the currently active step (0-based)

orientation | 'horizontal' | 'vertical' | 'horizontal' | Layout orientation of the stepper

showNavigation | boolean | true | Whether to show navigation elements

interactive | boolean | true | Whether steps are clickable for navigation

size | 'xs' | 'sm' | 'md' | 'lg' | 'md' | Visual size of the stepper component

disabledSteps | number[] | [] | Array of step indices that should be disabled

iconColor | string | '#fff' | Color for the check mark icons in completed steps

iconSize | string | '20' | Size of the check mark icons

stepVariant | 'default' | 'icon' | 'default' | Variant of step display (default numbers or icons)

showLabel | boolean | true | Whether to show step labels below circles

customStyles | Record<string, string> | {} | Custom CSS styles to apply to the stepper

[/TABLE]

Outputs

[TABLE]

Event | Type | Description

stepChange | EventEmitter<number> | Emitted when user navigates to a different step

stepperComplete | EventEmitter<void> | Emitted when the workflow reaches completion

[/TABLE]

Methods

[TABLE]

Method | Parameters | Return Type | Description

goToStep(index: number) | index: number | void | Navigate to a specific step programmatically

isDisabled(index: number) | index: number | boolean | Check if a specific step is disabled

[/TABLE]

CSS Custom Properties

[TABLE]

Property | Default | Description

--stepper-wrapper-background | Dynamic | Background color for active/completed elements

--stepper-background | Dynamic | Background color for inactive elements

--stepper-line-completed-background | Dynamic | Background for completed progress lines

--step-label-font | Dynamic | Font size for step labels

--step-label-active-font-weight | Dynamic | Font weight for active/completed labels

--step-circle-text | Dynamic | Text color for step circles

--stepper-size-sm-circle-size | Dynamic | Circle size for small variant

--stepper-size-md-circle-size | Dynamic | Circle size for medium variant

--stepper-size-lg-circle-size | Dynamic | Circle size for large variant

--stepper-size-sm-font | Dynamic | Font size for small variant

--stepper-size-md-font | Dynamic | Font size for medium variant

--stepper-size-lg-font | Dynamic | Font size for large variant

[/TABLE]

Accessibility Guidelines

Keyboard Navigation

- Tab: Navigate to stepper and move between interactive elements
- Enter: Activate focused step (if interactive)
- Space: Activate focused step (if interactive)
- Arrow Keys: Alternative navigation between steps

Screen Reader Support

- Use descriptive step labels that clearly indicate the purpose
- Provide context about the total number of steps
- Announce step changes and progress updates
- Include completion status in step descriptions
- Use appropriate heading levels for step content

Visual Design

- Maintain sufficient color contrast (4.5:1 minimum) for all states
- Provide clear focus indicators on interactive elements
- Ensure step circles meet minimum touch target size (44px)
- Use consistent visual hierarchy across step states
- Support high contrast and reduced motion preferences

Best Practices

Design Guidelines

- Use clear, action-oriented step labels
- Keep step labels concise but descriptive
- Provide visual feedback for all step states
- Consider orientation based on layout constraints
- Group related steps logically in the workflow

Performance

- Minimize step array modifications to prevent re-rendering
- Use OnPush change detection strategy for optimal performance
- Debounce rapid step changes if triggered programmatically
- Optimize animations for lower-end devices

SIDEBAR

The Sidebar component is a comprehensive navigation component designed to provide structured navigation, user management, and content organization in modern web applications. It features collapsible functionality, nested menu structures, user profiles, and integrated search capabilities.

The

```
<aava-sidebar>
```

component offers a flexible and responsive sidebar solution that can adapt to different screen sizes and user preferences, making it ideal for dashboards, admin panels, and complex navigation systems.

How to use

```
import
{
  AavaSidebarComponent
,
  MenuItem
,
  UserProfile
}
from
"@aava/play-core"
;
```

Basic Usage

Simple sidebar with basic navigation and user profile.

```
<div class="sidebar-demo">
<aava-sidebar
[width]="sidebarWidth()"
[collapsedWidth]="collapsedSidebarWidth()"
[showCollapseButton]="true"
[buttonVariant]=""inside""
[isCollapsed]="isCollapsed()">
```

```
[position]="'left'"  
(collapseToggle)="onCollapseToggle($event)"  
>  
<div class="demo-header-content">  
  <span class="header-title">My App</span>  
</div>  
<ul slot="content" class="demo-content">  
  <li  
    *ngFor="let item of menuItems"  
    [class.active]="item.active"  
    class="nav-item">  
    >  
    <span class="nav-icon">{{ item.icon }}</span>  
    <span *ngIf="!isCollapsed()">{{ item.label }}</span>  
  </li>  
</ul>  
</aava-sidebar>  
</div>  
---  
isCollapsed = signal<boolean>(false);  
sidebarWidth = signal<string>'280px';  
collapsedSidebarWidth = signal<string>'70px';  
menuItems = [  
  { icon: '■', label: 'Dashboard', active: true },  
  { icon: '■', label: 'Users' },  
  { icon: '■', label: 'Projects' },  
  { icon: '■', label: 'Analytics' },  
  { icon: '■■', label: 'Settings' },  
];
```

```
onCollapseToggle(collapsed: boolean): void {  
    this.isCollapsed.set(collapsed);  
}
```

Features

Responsive Design

- Automatic size adjustments based on screen dimensions
- Collapsible functionality for space optimization
- Mobile-friendly collapsed state

Flexible Layout

- Three size variants for different use cases
- Configurable header and footer sections
- Custom content projection slots

Advanced Navigation

- Nested sub-menu support with expandable sections
- Active state management for navigation items
- Icon support for visual navigation

User Experience

- Integrated search functionality
- User profile display and interaction
- Smooth collapse/expand animations

Customization

- Logo and branding options
- Theme-aware styling
- CSS custom properties for theming

API Reference

Input Properties

[TABLE]

Property | Type | Default | Description

width | string | '' | Sidebar width

collapsedWidth | string | '108px' | Collapsed sidebar width

height | string | '' | Sidebar height

hoverAreaWidth | string | '10px' | Hover area width

showCollapseButton | boolean | true | Show collapse/expand button

buttonVariant | 'inside' | 'outside' | 'inside' | Collapse button position

isCollapsed | boolean | false | Initial collapsed state

position | 'left' | 'right' | 'left' | Sidebar position

toggleOffset | number | 400 | Toggle button offset

togglePosition | 'top' | 'center' | 'bottom' | 'top' | Toggle button position

customStyles | Record<string, string> | {} | Custom CSS styles

[/TABLE]

Output Events

[TABLE]

Event | Type | Description

collapseToggle | EventEmitter<boolean> | Emitted when sidebar collapse state changes

[/TABLE]

Content Projection Slots

[TABLE]

Selector | Description

[slot=content] | Custom content above the navigation menu

[slot=footer] | Custom footer content above user profile

[/TABLE]

Best Practices

Menu Organization

- Logical Grouping: Group related menu items together
- Hierarchical Structure: Use sub-menus for complex navigation

- Consistent Icons: Use meaningful and consistent icons for menu items
- Clear Labels: Provide descriptive text for navigation items

Responsive Considerations

- Mobile-First: Design for mobile devices first
- Collapsed State: Ensure collapsed sidebar remains functional
- Touch Targets: Maintain adequate touch target sizes
- Content Priority: Prioritize essential navigation in collapsed state

User Experience

- Search Integration: Implement search for applications with many menu items
- Active States: Clearly indicate current navigation location
- Smooth Transitions: Use smooth animations for state changes
- Accessibility: Ensure keyboard navigation and screen reader support

Performance

- Lazy Loading: Load sub-menu content on demand
- Efficient Rendering: Use OnPush change detection strategy
- Memory Management: Clean up event listeners and subscriptions

Styling

The sidebar component uses CSS custom properties for theming:

CSS Custom Properties

[TABLE]

Property | Default Value | Description

Property	Default Value	Description
--sidebar-background	#ffffff	Background color of the sidebar
--sidebar-border	#e2e8f0	Border color of the sidebar

[/TABLE]

DRAWER

The

```
<aava-drawer>
```

component is a powerful sliding panel that provides a flexible overlay interface for navigation, forms, content display, and interactive elements. It supports multiple positions, sizes, animations, and advanced features like resizing and persistence, making it ideal for creating modern, accessible user interfaces.

How to use

```
import
{
  AavaDrawerComponent
}
from
"@aava/play-core"
;
```

Basic Usage

A simple drawer with default right position and medium size.

```
<aava-drawer
  [isOpen]="isDrawerOpen"
  title="Basic Drawer"
  subtitle="This is a simple drawer example"
  (opened)="onDrawerOpened()"
  (closed)="onDrawerClosed()"
  (closed)="closeDrawer()"
>
<div class="drawer-content">
  <h4>Welcome to the Drawer!</h4>
  <p>This is a basic drawer component with default settings:</p>
  <ul>
```

```
<li>Position: Right (default)</li>
<li>Size: Medium (default)</li>
<li>Overlay: Enabled</li>
<li>Animations: Enabled</li>
</ul>

<p>You can close this drawer by:</p>
<ul>
<li>Clicking the X button</li>
<li>Clicking the overlay</li>
<li>Pressing the Escape key</li>
</ul>
</div>

<div slot="footer">
<aava-button label="Close" variant="secondary" (click)="closeDrawer()">
</aava-button>
<aava-button label="Save" variant="primary" (click)="closeDrawer()">
</aava-button>
</div>
</aava-drawer>

---

isDrawerOpen = false;

openDrawer(): void {
  this.isDrawerOpen = true;
}

closeDrawer(): void {
  this.isDrawerOpen = false;
}

onDrawerOpened(): void {
  console.log('Drawer opened');
}
```

```
}
```

```
onDrawerClosed(): void {
```

```
    console.log('Drawer closed');
```

```
}
```

Positions

The drawer component supports four different positions for various use cases.

```
<aava-drawer
```

```
[isOpen]="rightDrawerOpen"
```

```
position="right"
```

```
title="Right Drawer"
```

```
(closed)="closeDrawer('right')"
```

```
>
```

```
<div class="drawer-content">
```

```
<h4>Right Position</h4>
```

```
<p>This drawer slides in from the right side.</p>
```

```
<p>This is the default position for drawers.</p>
```

```
<ul>
```

```
<li>Perfect for forms and detail panels</li>
```

```
<li>Common pattern for editing content</li>
```

```
<li>Good for mobile-first designs</li>
```

```
</ul>
```

```
</div>
```

```
</aava-drawer>
```

```
<!-- Left Position -->
```

```
<aava-drawer
```

```
[isOpen]="leftDrawerOpen"
```

```
position="left"
```

```
title="Left Drawer"
```

```
(closed)="closeDrawer('left')"
```

```
>

<div class="drawer-content">

<h4>Left Position</h4>

<p>This drawer slides in from the left side.</p>

<p>Perfect for navigation menus and sidebars.</p>

<ul>

<li>Traditional navigation pattern</li>

<li>Good for hierarchical menus</li>

<li>Familiar to users</li>

</ul>

</div>

</aava-drawer>

<!-- Top Position -->

<aava-drawer

[isOpen]="topDrawerOpen"

position="top"

title="Top Drawer"

(closed)="closeDrawer('top')"

>

<div class="drawer-content">

<h4>Top Position</h4>

<p>This drawer slides in from the top.</p>

<p>Great for notifications or quick actions.</p>

<ul>

<li>Perfect for notifications</li>

<li>Good for search interfaces</li>

<li>Quick access to tools</li>

</ul>
```

```
</div>

</aava-drawer>

<!-- Bottom Position -->

<aava-drawer
  [isOpen]="bottomDrawerOpen"
  position="bottom"
  title="Bottom Drawer"
  (closed)="closeDrawer('bottom')"

>
<div class="drawer-content">
  <h4>Bottom Position</h4>
  <p>This drawer slides in from the bottom.</p>
  <p>Ideal for mobile-first designs and action sheets.</p>
  <ul>
    <li>Mobile-friendly pattern</li>
    <li>Good for action sheets</li>
    <li>Thumb-friendly interaction</li>
  </ul>
</div>

</aava-drawer>

---

// Drawer states for different positions
rightDrawerOpen = false;
leftDrawerOpen = false;
topDrawerOpen = false;
bottomDrawerOpen = false;

openDrawer(position: string): void {
  switch (position) {
    case 'right':
```

```
this.rightDrawerOpen = true;  
break;  
  
case 'left':  
this.leftDrawerOpen = true;  
break;  
  
case 'top':  
this.topDrawerOpen = true;  
break;  
  
case 'bottom':  
this.bottomDrawerOpen = true;  
break;  
}  
}  
  
closeDrawer(position: string): void {  
switch (position) {  
  
case 'right':  
this.rightDrawerOpen = false;  
break;  
  
case 'left':  
this.leftDrawerOpen = false;  
break;  
  
case 'top':  
this.topDrawerOpen = false;  
break;  
  
case 'bottom':  
this.bottomDrawerOpen = false;  
break;  
}  
}
```

Sizes

Choose from five predefined sizes or use custom dimensions.

```
<aava-drawer  
[isOpen]="'smallDrawerOpen'"  
size="small"  
title="Small Drawer"  
(closed)="closeDrawer('small')"  
>  
<div class="drawer-content">  
<h4>Small Size (320px)</h4>  
<p>This is a small drawer perfect for simple forms or quick actions.</p>  
<ul>  
<li>Width: 320px</li>  
<li>Height: 200px (for top/bottom)</li>  
<li>Perfect for simple forms</li>  
<li>Quick actions and notifications</li>  
</ul>  
</div>  
</aava-drawer>  
<!-- Medium Size (Default) -->  
<aava-drawer  
[isOpen]="'mediumDrawerOpen'"  
size="medium"  
title="Medium Drawer"  
(closed)="closeDrawer('medium')"  
>  
<div class="drawer-content">  
<h4>Medium Size (480px)</h4>  
<p>This is a medium drawer - the default size, good for most use cases.</p>
```

```
<ul>
<li>Width: 480px</li>
<li>Height: 300px (for top/bottom)</li>
<li>Default size</li>
<li>Good for most use cases</li>
</ul>
</div>
</aava-drawer>
<!-- Large Size -->
<aava-drawer
[isOpen] ="largeDrawerOpen"
size="large"
title="Large Drawer"
(closed) ="closeDrawer('large')"
>
<div class="drawer-content">
<h4>Large Size (640px)</h4>
<p>This is a large drawer great for detailed forms or content.</p>
<ul>
<li>Width: 640px</li>
<li>Height: 400px (for top/bottom)</li>
<li>Great for detailed forms</li>
<li>Content-heavy interfaces</li>
</ul>
</div>
</aava-drawer>
<!-- Extra Large Size -->
<aava-drawer
[isOpen] ="extraLargeDrawerOpen"
```

```
size="extra-large"
title="Extra Large Drawer"
(closed)="closeDrawer('extra-large')"
>
<div class="drawer-content">
<h4>Extra Large Size (800px)</h4>
<p>
This is an extra large drawer for complex interfaces or detailed views.
</p>
<ul>
<li>Width: 800px</li>
<li>Height: 500px (for top/bottom)</li>
<li>Complex interfaces</li>
<li>Detailed views and dashboards</li>
</ul>
</div>
</aava-drawer>
<!-- Full Size -->
<aava-drawer
[isOpen]="fullDrawerOpen"
size="full"
title="Full Size Drawer"
(closed)="closeDrawer('full')"
>
<div class="drawer-content">
<h4>Full Size (100% viewport)</h4>
<p>
This drawer takes up the full viewport - essentially a full-screen modal
experience.

```

```
</p>

<ul>
  <li>Width: 100% viewport</li>
  <li>Height: 100% viewport</li>
  <li>Full-screen modal experience</li>
  <li>Immersive content display</li>
</ul>
</div>

</aava-drawer>

---

// Drawer states for different sizes

smallDrawerOpen = false;

mediumDrawerOpen = false;

largeDrawerOpen = false;

extraLargeDrawerOpen = false;

fullDrawerOpen = false;

openDrawer(size: string): void {
  switch (size) {
    case 'small':
      this.smallDrawerOpen = true;
      break;
    case 'medium':
      this.mediumDrawerOpen = true;
      break;
    case 'large':
      this.largeDrawerOpen = true;
      break;
    case 'extra-large':
      this.extraLargeDrawerOpen = true;
  }
}
```

```
break;

case 'full':
  this.fullDrawerOpen = true;
break;
}

}

closeDrawer(size: string): void {
  switch (size) {
    case 'small':
      this.smallDrawerOpen = false;
      break;
    case 'medium':
      this.mediumDrawerOpen = false;
      break;
    case 'large':
      this.largeDrawerOpen = false;
      break;
    case 'extra-large':
      this.extraLargeDrawerOpen = false;
      break;
    case 'full':
      this.fullDrawerOpen = false;
      break;
  }
}
```

Content Structure

Organize drawer content with header, body, and footer sections.

[Error extracting code]

Features

Multiple Positions

- Right: Default position, slides in from the right edge
- Left: Slides in from the left edge, perfect for navigation
- Top: Slides down from the top, ideal for notifications
- Bottom: Slides up from the bottom, great for forms

Size Variants

- Small: 320px width (200px height for top/bottom)
- Medium: 480px width (300px height for top/bottom) - Default
- Large: 640px width (400px height for top/bottom)
- Extra Large: 800px width (500px height for top/bottom)
- Full: 100% viewport width/height
- Custom: Specify exact dimensions with width/height properties

Advanced Features

- Resizable: Enable drag-to-resize functionality
- Persistent: Prevent accidental closing
- Custom Animations: Spring-based animations with reduced motion support
- Overlay Control: Configurable overlay behavior
- Keyboard Navigation: Full keyboard accessibility
- Focus Management: Proper focus trapping and restoration

Accessibility

- ARIA Support: Complete ARIA attributes and roles
- Screen Reader: Full screen reader compatibility
- Keyboard Navigation: Escape key, tab trapping
- Reduced Motion: Respects user motion preferences
- High Contrast: High contrast mode support

API Reference

Inputs

[TABLE]

Property | Type | Default | Description

isOpen | boolean | false | Controls the visibility of the drawer

position | DrawerPosition | 'right' | Position of the drawer on screen

size | DrawerSize | 'medium' | Predefined size of the drawer

showOverlay | boolean | true | Whether to show the backdrop overlay

closeOnOverlayClick | boolean | true | Close drawer when overlay is clicked

closeOnEscape | boolean | true | Close drawer when Escape key is pressed

showCloseButton | boolean | true | Show the close button in header

persistent | boolean | false | Prevent drawer from being closed

resizable | boolean | false | Enable resize functionality

animate | boolean | true | Enable animations

title | string | '' | Title displayed in the header

subtitle | string | '' | Subtitle displayed in the header

showHeader | boolean | true | Show the header section

showFooter | boolean | false | Show the footer section

width | string | '' | Custom width (overrides size)

height | string | '' | Custom height (overrides size)

maxWidth | string | '' | Maximum width constraint

maxHeight | string | '' | Maximum height constraint

zIndex | number | 1050 | Z-index for drawer positioning

closeIcon | string | 'X' | Icon name for close button

closeIconSize | number | 20 | Size of the close icon

[/TABLE]

Outputs

[TABLE]

Event | Type | Description

opened | EventEmitter<void> | Emitted when drawer opens

closed | EventEmitter<void> | Emitted when drawer closes
overlayClick | EventEmitter<void> | Emitted when overlay is clicked
escapePressed | EventEmitter<void> | Emitted when Escape key is pressed
[/TABLE]

Methods

[TABLE]

Method	Parameters	Return Type	Description
open()	None	void	Opens the drawer
close()	None	void	Closes the drawer
toggle()	None	void	Toggles drawer open/closed state
onOverlayClick()	None	void	Handles overlay click event
onCloseClick()	None	void	Handles close button click
onDrawerClick()	Event	void	Prevents event bubbling
getDrawerClasses()	None	string	Returns CSS classes for drawer
getOverlayClasses()	None	string	Returns CSS classes for overlay
getDrawerStyles()	None	object	Returns inline styles for drawer

[/TABLE]

Types

type

DrawerPosition

=

"left"

|

"right"

|

"top"

|

"bottom"

;

```
type  
DrawerSize  
=  
"small"  
|  
"medium"  
|  
"large"  
|  
"extra-large"  
|  
"full"  
;
```

Content Projection

The drawer component supports content projection with specific slots:

[TABLE]

Slot Name	Description
[slot=header]	Custom content for the header section
[slot=footer]	Custom content for the footer section
Default	Main content displayed in the body section

[/TABLE]

CSS Classes

The component provides several CSS classes for styling:

[TABLE]

Class Name	Description
.ava-drawer	Main drawer container
.ava-drawer--open	Applied when drawer is open
.ava-drawer--left	Left position styling

.ava-drawer--right | Right position styling
.ava-drawer--top | Top position styling
.ava-drawer--bottom | Bottom position styling
.ava-drawer--small | Small size styling
.ava-drawer--medium | Medium size styling
.ava-drawer--large | Large size styling
.ava-drawer--extra-large | Extra large size styling
.ava-drawer--full | Full size styling
.ava-drawer--resizable | Resizable functionality styling
.ava-drawer-overlay | Overlay container
.ava-drawer-overlay--open | Applied when overlay is visible
.ava-drawer__animation-wrapper | Animation wrapper
.ava-drawer__content | Content container
.ava-drawer__header | Header section
.ava-drawer__header-content | Header content area
.ava-drawer__title-section | Title and subtitle container
.ava-drawer__title | Title element
.ava-drawer__subtitle | Subtitle element
.ava-drawer__header-slot | Header slot container
.ava-drawer__close-section | Close button container
.ava-drawer__body | Main content area
.ava-drawer__footer | Footer section
.ava-drawer__resize-handle | Resize handle

[/TABLE]

CSS Custom Properties

The component uses CSS custom properties for theming:

[TABLE]

Property | Description

--drawer-background | Background color of the drawer

--drawer-border | Border styling for the drawer
--drawer-shadow | Box shadow for the drawer
--drawer-z-index | Z-index for drawer positioning
--drawer-spring-duration | Animation duration (300ms)
--drawer-spring-easing | Animation easing function
--drawer-header-padding | Header padding
--drawer-header-border | Header bottom border
--drawer-header-background | Header background color
--drawer-header-gap | Gap between header elements
--drawer-title-font | Title font styling
--drawer-title-color | Title text color
--drawer-title-weight | Title font weight
--drawer-title-line-height | Title line height
--drawer-subtitle-font | Subtitle font styling
--drawer-subtitle-color | Subtitle text color
--drawer-subtitle-weight | Subtitle font weight
--drawer-subtitle-line-height | Subtitle line height
--drawer-subtitle-margin | Subtitle margin
--drawer-body-padding | Body padding
--drawer-body-background | Body background color
--drawer-body-text-color | Body text color
--drawer-footer-padding | Footer padding
--drawer-footer-border | Footer top border
--drawer-footer-background | Footer background color
--popup-overlay-background | Overlay background color
--global-motion-duration-standard | Standard motion duration
--global-motion-easing-standard | Standard motion easing
--global-spacing-2 | Small spacing token

--global-spacing-3 | Medium spacing token

[/TABLE]

Best Practices

User Experience

- Appropriate Size: Choose drawer size based on content complexity
- Clear Purpose: Use descriptive titles and content structure
- Smooth Interactions: Enable animations for better UX
- Consistent Behavior: Maintain consistent drawer patterns across the app
- Responsive Design: Ensure drawers work well on all screen sizes

Performance

- Conditional Rendering: Only render drawer content when needed
- Animation Optimization: Use hardware-accelerated animations
- Memory Management: Clean up event listeners and timeouts
- Change Detection: Leverage OnPush strategy for better performance
- Content Loading: Load heavy content after drawer opens

Accessibility

- Keyboard Navigation: Ensure all interactions work with keyboard
- Screen Reader Support: Provide proper ARIA labels and descriptions
- Focus Management: Trap focus within drawer when open
- Motion Preferences: Respect user's reduced motion preferences
- High Contrast: Ensure visibility in high contrast mode

Content Organization

- Header Structure: Use clear titles and optional subtitles
- Content Hierarchy: Organize content logically within body
- Footer Actions: Place primary actions in footer when appropriate
- Scrolling: Ensure long content scrolls properly within body
- Responsive Content: Adapt content layout for different drawer sizes

Accessibility Guidelines

Screen Reader Support

- ARIA Attributes: Proper role="dialog", aria-modal="true"
- Labels: aria-labelledby and aria-describedby for content identification
- State Announcements: Clear announcements for drawer state changes
- Content Structure: Semantic HTML structure for better navigation

Keyboard Navigation

- Tab Order: Logical tab order within the drawer
- Escape Key: Close drawer with Escape key (configurable)
- Focus Trapping: Focus remains within drawer when open
- Focus Restoration: Return focus to trigger element when closed

Visual Design

- High Contrast: Ensure sufficient contrast ratios
- Focus Indicators: Clear focus indicators for all interactive elements
- Color Independence: Don't rely solely on color for information
- Text Scaling: Support for text scaling and zoom

Motion and Animation

- Reduced Motion: Respect prefers-reduced-motion media query
- Animation Duration: Keep animations smooth but not distracting
- Motion Alternatives: Provide alternatives for users who can't see animations
- Performance: Ensure animations don't cause performance issues

Responsive Behavior

Mobile Adaptations

The drawer component automatically adapts to mobile screens:

- Full Width/Height: Drawers become full viewport on mobile
- Touch Optimization: Optimized for touch interactions
- Viewport Adaptation: Adapts to different mobile viewport sizes
- Performance: Optimized performance for mobile devices

Breakpoint Behavior

- Desktop (>768px): Full drawer with all features
- Mobile ($\leq 768\text{px}$): Optimized drawer for mobile screens
- Content Scaling: Drawer content scales appropriately
- Animation Performance: Optimized animations for different devices

Content Considerations

- Flexible Sizing: Drawer adapts to different content sizes
- Layout Preservation: Maintains layout consistency across devices
- Loading States: Consistent loading experience across platforms
- Performance: Efficient rendering on all device types

TOOLTIP

A flexible and accessible tooltip component for providing contextual information, guidance, or feedback. Supports multiple positions, arrow alignments, semantic variants, smooth animations, and advanced content, ensuring both usability and accessibility.

How to use

```
import  
{  
  AavaTooltipDirective  
}  
from  
"@aava/play-core"  
;
```

Basic Usage

A simple tooltip attached to any element.

```
<aava-button  
  label="Tooltip with Hover"  
  variant="primary"  
  size="small"  
  state="default"  
  avaTooltipDescription="Tooltip with Hover"  
  avaTooltipType="simple"  
  avaTooltipTrigger="hover"  
  avaTooltipPosition="top"  
  avaTooltipArrow="center"  
>  
</aava-button>
```

Positions

Tooltips can be positioned relative to the target element:

```
top  
(default),  
bottom  
,  
left  
,  
right  
. Arrow alignment options:  
start  
,  
center  
(default),  
end  
. .  
<aava-button  
label=" Start"  
variant="primary"  
size="small"  
state="default"  
avaTooltipDescription=" Start"  
avaTooltipType="simple"  
avaTooltipTrigger="hover"  
avaTooltipPosition="top"  
avaTooltipArrow="start"  
></aava-button>  
<aava-button  
label=" Center"  
variant="primary"  
size="small"
```

```
state="default"
avaTooltipDescription=" Center"
avaTooltipType="simple"
avaTooltipTrigger="hover"
avaTooltipPosition="top"
avaTooltipArrow="center"
></aava-button>
<aava-button
label=" End"
variant="primary"
size="small"
state="default"
avaTooltipDescription=" End"
avaTooltipType="simple"
avaTooltipTrigger="hover"
avaTooltipPosition="top"
avaTooltipArrow="end"
></aava-button>
```

Variants

```
<aava-button
label="Title Only"
variant="secondary"
size="small"
avaTooltipTitle="Heading"
avaTooltipPosition="top"
avaTooltipArrow="center"
>
</aava-button>
<aava-button
```

```
label="Desc Only"
variant="secondary"
size="small"
avaTooltipDescription="This description should start from beginning"
avaTooltipPosition="top"
avaTooltipArrow="center"
>
</aava-button>
<aava-button
label="Desc+Icon Only"
variant="secondary"
size="small"
avaTooltipDescription="This description should start from beginning"
avaToolipIcon="info"
avaToolipIconColor="var(--color-text-primary)"
avaTooltipPosition="top"
avaTooltipArrow="center"
>
</aava-button>
<aava-button
label="Title+Icon+Desc"
variant="secondary"
size="small"
avaTooltipTitle="Save Feature"
avaTooltipDescription="This description should align with title text"
avaToolipIcon="save"
avaToolipIconColor="var(--color-text-primary)"
avaTooltipPosition="top"
```

```
avaTooltipArrow="center"  
>  
</aava-button>
```

API Reference

Inputs (viaconfigobject)

[TABLE]

Property | Type | Default | Description

content | string | "" | Tooltip text or HTML content

position | 'top' | 'bottom' | 'left' | 'right' | 'top' | Tooltip position relative to target

arrow | 'start' | 'center' | 'end' | 'center' | Arrow alignment

width | number | - | Target element width (for positioning, set automatically)

height | number | - | Target element height (for positioning, set automatically)

top | number | - | Target element top offset (set automatically)

left | number | - | Target element left offset (set automatically)

bottom | number | - | Target element bottom offset (set automatically)

enableAnimation | boolean | true | Enable/disable tooltip animation

behavior | 'hover' | 'focus' | 'manual' | 'hover' | How the tooltip is triggered

[/TABLE]

Outputs

Tooltip is a presentational component. Events are handled by the directive or parent logic.

Methods

[TABLE]

Method | Parameters | Description

updatePosition() | - | Recalculates and updates tooltip position

[/TABLE]

CSS Custom Properties

Tooltip uses CSS custom properties for theming and design tokens:

[TABLE]

Property | Description

--tooltip-background | Tooltip background color
--tooltip-text | Tooltip text color
--tooltip-font | Font family for tooltip
--tooltip-padding | Padding inside tooltip
--tooltip-border-radius | Border radius for tooltip
--tooltip-shadow | Box shadow for tooltip
--tooltip-border-color | Border color for tooltip
--tooltip-z-index | Z-index for tooltip
--tooltip-info-background | Info variant background
--tooltip-info-text | Info variant text color
--tooltip-info-arrow | Info variant arrow color
--tooltip-success-background | Success variant background
--tooltip-success-text | Success variant text color
--tooltip-success-arrow | Success variant arrow color
--tooltip-warning-background | Warning variant background
--tooltip-warning-text | Warning variant text color
--tooltip-warning-arrow | Warning variant arrow color
--tooltip-error-background | Error variant background
--tooltip-error-text | Error variant text color
--tooltip-error-arrow | Error variant arrow color

[/TABLE]

Accessibility

Tooltip follows accessibility best practices:

- Keyboard accessible (triggered by focus-hover)
- Proper ARIA attributes for screen readers
- Focus management and visible indicators
- High contrast and reduced motion support

Theming & Design Tokens

All colors, spacing, and effects are controlled via semantic design tokens and CSS custom properties. Override these in your theme or component styles for custom branding.

Best Practices

- Use tooltips for short, contextual information only
- Avoid placing critical information solely in tooltips
- Ensure tooltips are accessible via keyboard and screen readers
- Use semantic variants to match the context (info, warning, etc.)
- Test tooltip placement in responsive layouts
- Avoid excessive custom styling; use built-in variants and sizes

POPUP

The

```
<ava-popup>
```

component provides a highly flexible floating container for displaying contextual content such as tooltips, menus, dropdowns, or custom overlays. It supports multiple trigger types, advanced positioning, animation, and full accessibility compliance. Use it to create interactive overlays that appear on click, hover, focus, or programmatically.

How to use

```
import  
{  
  PopupComponent  
}  
from  
"@aava/play-core"  
;
```

Basic Usage

A simple popup triggered by a button click, displaying custom content.

```
// app.component.html  
  
<button aava-button (click)="isOpen = !isOpen">Toggle Popup</button>  
  
<aava-popup [isOpen]="isOpen" (closed)="isOpen = false">  
  <div style="padding: 1rem;">Hello from Popup!</div>  
</aava-popup>
```

Positioning

Popup supports multiple placement options relative to the trigger element:

- Top
- Bottom
- Left
- Right
- Auto (smart positioning)

```
// app.component.html

<button aava-button>Top</button>

<aava-popup placement="top">
<div style="padding: 1rem;">Top Placement</div>
</aava-popup>

<button aava-button>Bottom</button>

<aava-popup placement="bottom">
<div style="padding: 1rem;">Bottom Placement</div>
</aava-popup>

<button aava-button>Left</button>

<aava-popup placement="left">
<div style="padding: 1rem;">Left Placement</div>
</aava-popup>

<button aava-button>Right</button>

<aava-popup placement="right">
<div style="padding: 1rem;">Right Placement</div>
</aava-popup>

<button aava-button>Auto</button>

<aava-popup placement="auto">
<div style="padding: 1rem;">Auto Placement</div>
</aava-popup>
```

Custom Content

Render any custom content inside the popup, including forms, lists, or interactive elements.

```
// app.component.html

<button aava-button>Show Custom Content</button>

<aava-popup>
<form style="padding: 1rem;">
<label>Name: <input type="text" /></label><br />
<button type="submit" aava-button>Submit</button>
```

```
</form>  
</aava-popup>
```

Accessibility

Popup is fully accessible:

- Keyboard navigation
- ARIA roles and attributes
- Focus management
- Dismiss on Escape

API Reference

Inputs

[TABLE]

Property | Type | Default | Description

isOpen | boolean | false | Whether the popup is open

trigger | 'click' | 'hover' | 'focus' | 'manual' | 'click' | Trigger type for showing the popup

placement | 'top' | 'bottom' | 'left' | 'right' | 'auto' | 'bottom' | Popup placement relative to trigger

offset | number | 8 | Offset distance from trigger (px)

closeOnClickOutside | boolean | true | Close popup when clicking outside

closeOnEscape | boolean | true | Close popup on Escape key

disabled | boolean | false | Disable the popup

backdrop | boolean | false | Show a backdrop behind the popup

zIndex | number | 1000 | z-index for popup layering

animation | 'fade' | 'scale' | 'none' | 'fade' | Animation style for popup

containerClass | string | '' | Custom CSS class for popup container

containerStyle | Record<string, string> | {} | Custom styles for popup container

[/TABLE]

Outputs

[TABLE]

Event	Type	Description
opened	EventEmitter<void>	Emitted when popup is opened
closed	EventEmitter<void>	Emitted when popup is closed
positioned	EventEmitter<string>	Emitted when popup is positioned

[/TABLE]

Methods

[TABLE]

Method	Parameters	Description
open()	void	Open the popup programmatically
close()	void	Close the popup programmatically
toggle()	void	Toggle popup open/close state
reposition()	void	Recalculate popup position

[/TABLE]

CSS Custom Properties

[TABLE]

Property	Description
--popup-background	Background color of the popup
--popup-border-radius	Border radius of the popup
--popup-box-shadow	Box shadow for the popup
--popup-z-index	z-index for popup layering
--popup-animation-duration	Duration of the popup animation
--popup-arrow-size	Size of the popup arrow (if present)
--popup-arrow-color	Color of the popup arrow
--popup-backdrop-background	Background color for the backdrop

[/TABLE]

Accessibility Guidelines

- Keyboard Navigation: Tab, Shift+Tab, Escape to close
- ARIA Roles: role="dialog" or role="tooltip" as appropriate

- Focus Management: Focus is trapped within popup when open
- Dismissal: Popup closes on Escape or outside click
- Screen Reader Support: Popup content is announced

Best Practices

- Use the appropriate trigger for your use case (click for menus, hover for tooltips)
- Always provide accessible labels and ARIA attributes
- Avoid using popups for critical information that must not be missed
- Test keyboard and screen reader interactions
- Use smart positioning to avoid clipping or overflow
- Keep popup content concise and focused

TOAST

The

```
<ava-toast>
```

system provides a comprehensive notification solution with multiple variants, flexible positioning, smooth animations, and action buttons. Built around a service-based architecture, it offers both simple and advanced toast configurations for various user feedback scenarios.

How to use

```
import
{
    AavaToastService
}
from
"@aava/play-core"
;

// Inject the service
constructor
(
    private
        toastService
    :
    AavaToastService
)
{
}

// Show different types of toasts
this
.

toastService
```

```
success
(
{
  message
  :
  'Operation completed!'
}
)
;
this

.

toastService

.

error
(
{
  message
  :
  'Something went wrong!'
}
)
;
```

Basic Usage

Simple toast notifications with default configurations.

```
<aava-button
  label="Show Success Toast"
  variant="success"
  (userClick)="showSuccessToast()"
></aava-button>
```

```
--  
// Inject the toast service in your component  
  
constructor(private toastService: ToastService) {}  
  
// Show different types of toasts  
  
showSuccessToast() {  
  
  this.toastService.success({  
  
    title: 'Successfully created!',  
  
    message: 'Your changes have been saved successfully',  
  
    duration: 2000,  
  
    customWidth: '400px',  
  
    design: 'modern',  
  
    size: 'large',  
  
  });  
  
}  
}
```

Variants

Six distinct toast variants for different notification types and user feedback scenarios.

```
<aava-button  
  label="Show Success Toast"  
  variant="success"  
  (userClick)="showSuccessToast()"></aava-button>  
  
<aava-button  
  label="Show Error Toast"  
  variant="danger"  
  (userClick)="showErrorToast()"></aava-button>  
  
<aava-button  
  label="Show Warning Toast"
```

```
variant="warning"
(userClick)="showWarningToast()"
></aava-button>
<aava-button
label="Show Info Toast"
variant="info"
(userClick)="showInfoToast()"
></aava-button>
<aava-button
label="Show Default Toast"
variant="secondary"
(userClick)="showDefaultToast()"
></aava-button>
---
// Inject the toast service in your component
constructor(private toastService: ToastService) {}
// Show different toast variants
showSuccessToast() {
  this.toastService.success({
    title: 'Successfully created!',
    message: 'Your changes have been saved successfully',
    duration: 2000,
    customWidth: '300px',
    design: 'modern',
    size: 'small',
  });
}
showErrorToast() {
  this.toastService.error({

```

```
title: 'Error Occurred',  
message: 'Connection error. Unable to connect to the server at present',  
duration: 2000,  
customWidth: '300px',  
design: 'modern',  
size: 'small',  
});  
}  
  
showWarningToast() {  
this.toastService.warning({  
title: 'Warning Occurred',  
message: 'Please review your input carefully before proceeding.',  
duration: 2000,  
customWidth: '300px',  
design: 'modern',  
size: 'small',  
});  
}  
  
showInfoToast() {  
this.toastService.info({  
title: 'Action Required',  
message: 'Please review your input carefully before proceeding.',  
duration: 2000,  
customWidth: '300px',  
design: 'modern',  
size: 'small',  
});  
}  
  
showDefaultToast() {
```

```

this.toastService.default({
  title: 'Default Toast Occurred',
  message: 'This is a default toast with neutral styling.',
  duration: 2000,
  customWidth: '300px',
  design: 'modern',
  size: 'small',
});

}

showCustomToast() {
  this.toastService.custom({
    title: 'Custom Toast',
    message: 'This is a fully customizable toast with unique styling.',
    customBackground: 'var(--color-primary)',
    customTextColor: 'var(--color-text-primary)',
    customWidth: '300px',
    design: 'modern',
    size: 'small',
  });
}

```

Available Variants

- Success- Green styling for successful operations and confirmations
- Error- Red styling for errors and critical issues
- Warning- Orange/yellow styling for warnings and cautions
- Info- Blue styling for informational messages and tips
- Default- Neutral styling for general notifications
- Custom- Fully customizable styling and content

Sizes

Three size variants to accommodate different content lengths and interface requirements.

```
<aava-button
    label="Show Small Success"
    variant="success"
    size="small"
    (userClick)="showSmallSuccess()"

></aava-button>

<aava-button
    label="Show Medium Warning"
    variant="warning"
    size="small"
    (userClick)="showMediumWarning()"

></aava-button>

<aava-button
    label="Show Large Success"
    variant="danger"
    size="small"
    (userClick)="showLargeError()"

></aava-button>

---

// Inject the toast service in your component
constructor(private toastService: ToastService) {}

// Show different toast sizes

showSmallSuccess() {
    this.toastService.success({
        title: 'Successfully created!',
        message: 'Your changes have been saved successfully',
        duration: 1000,
        customWidth: '300px',
    })
}
```

```

    design: 'modern',
    size: 'small',
  });
}

showMediumWarning() {
  this.toastService.warning({
    title: 'Action Required',
    message: 'Incomplete fields. Please fill in all required information.',
    duration: 1000,
    customWidth: '350px',
    design: 'modern',
    size: 'medium',
  });
}

showLargeError() {
  this.toastService.error({
    title: 'Error Occurred',
    message: 'Connection error. Unable to connect to the server at present',
    duration: 1000,
    customWidth: '400px',
    design: 'modern',
    size: 'large',
  });
}

```

Available Sizes

- Small- Compact size for minimal content and dense interfaces
- Medium- Standard size for most notification scenarios (default)
- Large- Prominent size for important messages and detailed content

Service Methods

The ToastService provides convenient methods for different toast types:

Success Toast

this

.

toastService

.

success

(

{

title

:

"Success!"

,

message

:

"Operation completed successfully."

,

duration

:

4000

,

showCloseButton

:

true

,

}

)

;

Error Toast

this

toastService

error

(

{

title

:

"Error!"

,

message

:

"Something went wrong. Please try again."

,

showRetryButton

:

true

,

retryButtonText

:

"Retry"

,

}

)

;

Warning Toast

```
this  
.  
toastService  
.  
warning  
(  
{  
title  
:  
"  
Warning"  
,  
message  
:  
"  
Please review the information carefully."  
,  
showActionButton  
:  
true  
,  
actionButtonText  
:  
"  
Review"  
,  
}  
)  
;  
;
```

Info Toast

```
this  
.  
;
```

```
toastService

    .
    .
    .
    info
    (
    {
        title
        :
        "Information"
        ,
        message
        :
        "Here is some important information."
        ,
        showLearnMoreButton
        :
        true
        ,
        learnMoreButtonText
        :
        "Learn More"
        ,
    }
    )
    ;

```

Custom Toast

```
this

    .
    .
    .
    toastService
```

```
custom
(
{
title
:
"Custom Toast"
,
message
:
"This is a fully customizable toast."
,
customWidth
:
"500px"
,
customBackground
:
"#6366f1"
,
customTextColor
:
"#ffffff"
,
}
)
;
```

API Reference

ToastService Methods

[TABLE]

Method | Parameters | Return Type | Description

success() | SuccessToastConfig? | Promise<ToastResult> | Show success toast with green styling

error() | ErrorToastConfig? | Promise<ToastResult> | Show error toast with red styling

warning() | WarningToastConfig? | Promise<ToastResult> | Show warning toast with orange styling

info() | InfoToastConfig? | Promise<ToastResult> | Show info toast with blue styling

default() | DefaultToastConfig? | Promise<ToastResult> | Show default toast with neutral styling

custom() | CustomToastConfig? | Promise<ToastResult> | Show custom toast with full customization

setPosition() | ToastPosition | void | Set global toast position

dismissAll() | - | void | Dismiss all active toasts

[/TABLE]

Toast Configuration Interfaces

[TABLE]

Property | Type | Default | Description

title | string? | " | Toast header text

message | string? | " | Toast body text

duration | number? | 4000 | Auto-dismiss duration in milliseconds

position | ToastPosition? | 'top-right' | Toast position on screen

showCloseButton | boolean? | true | Whether to show close button

showProgress | boolean? | true | Whether to show progress bar

icon | string? | " | Custom icon name

iconColor | string? | " | Custom icon color

customWidth | string? | " | Custom toast width

customHeight | string? | " | Custom toast height

design | 'classic' | 'modern' | 'classic' | Toast design variant

size | 'large' | 'medium' | 'small' | 'large' | Toast size variant

[/TABLE]

[TABLE]

Property | Type | Default | Description

type | 'success' | 'success' | Toast type (always 'success')

[/TABLE]

[TABLE]

Property | Type | Default | Description

type | 'error' | 'error' | Toast type (always 'error')

showRetryButton | boolean? | false | Whether to show retry button

retryButtonText | string? | 'Retry' | Text for retry button

[/TABLE]

[TABLE]

Property | Type | Default | Description

type | 'warning' | 'warning' | Toast type (always 'warning')

showActionButton | boolean? | false | Whether to show action button

actionButtonText | string? | 'Action' | Text for action button

[/TABLE]

[TABLE]

Property | Type | Default | Description

type | 'info' | 'info' | Toast type (always 'info')

showLearnMoreButton | boolean? | false | Whether to show learn more button

learnMoreButtonText | string? | 'Learn More' | Text for learn more button

[/TABLE]

[TABLE]

Property | Type | Default | Description

type | 'custom' | 'custom' | Toast type (always 'custom')

customContent | string? | "" | Custom HTML content

customBackground | string? | "" | Custom background color

customTextColor | string? | "" | Custom text color

progressColor | string? | "" | Custom progress bar color

showCustomActions | boolean? | false | Whether to show custom action buttons

customActions | CustomAction[]? | "" | Array of custom action buttons

[/TABLE]

Toast Position Options

[TABLE]

Position | Description

'top-left' | Top-left corner of the screen

'top-center' | Top-center of the screen

'top-right' | Top-right corner of the screen (default)

bottom-left | Bottom-left corner of the screen

bottom-center | Bottom-center of the screen

bottom-right | Bottom-right corner of the screen

[/TABLE]

Toast Result Interface

[TABLE]

Property | Type | Description

action | 'close' | 'retry' | 'action' | 'learn-more' | 'timeout' | string | Action that triggered toast dismissal

data | any? | Additional data from custom actions

[/TABLE]

Custom Action Interface

[TABLE]

Property | Type | Description

action | string | Unique identifier for the action

label | string | Display text for the action button

data | any? | Additional data to pass with the action

[/TABLE]

Best Practices

Design Guidelines

- Choose appropriate variants- Use semantic colors that match the message type
- Keep messages concise- Toast notifications should be brief and actionable

- Use consistent positioning- Stick to one position for your application
- Consider duration carefully- Longer durations for important messages, shorter for confirmations
- Limit concurrent toasts- Avoid overwhelming users with too many notifications

Accessibility

- Provide meaningful titles- Use descriptive headers for screen readers
- Include action alternatives- Ensure all toast actions are keyboard accessible
- Test with screen readers- Verify proper announcement of toast content
- Maintain focus management- Ensure focus returns to appropriate elements
- Use sufficient contrast- Maintain readability in both light and dark themes

Performance

- Limit toast instances- Avoid creating excessive toast components
- Use appropriate timeouts- Set reasonable auto-dismiss durations
- Clean up resources- Ensure proper cleanup of event listeners
- Optimize animations- Use CSS transforms for smooth performance

User Experience

- Position strategically- Choose positions that don't block important content
- Provide clear actions- Make action buttons descriptive and intuitive
- Handle errors gracefully- Use error toasts with retry options when appropriate
- Consider mobile users- Ensure toasts are readable on small screens
- Respect user preferences- Allow users to control toast behavior

Use Cases

- Success Confirmations- Operation completion, form submissions
- Error Notifications- Validation failures, API errors, network issues
- Warning Messages- Important reminders, confirmation requirements
- Information Updates- System status, feature announcements
- Action Prompts- User decisions, next steps, navigation hints

Technical Notes

Service Architecture

The toast system uses a service-based architecture where:

- ToastService manages all toast operations and lifecycle
- ToastContainerComponent provides the positioning and container logic
- Individual toast components handle specific styling and behavior
- Dynamic component creation enables programmatic toast generation

Positioning System

Toasts are positioned using CSS transforms and positioning:

- Fixed positioning ensures toasts appear above all content
- Responsive behavior adapts to different screen sizes
- Z-index management maintains proper layering
- Mobile optimization provides full-width toasts on small screens

Animation System

Toast animations include:

- Entrance animations- Slide and fade in effects
- Exit animations- Slide and fade out effects
- Progress bars- Visual countdown for auto-dismiss
- Hover interactions- Timer pause/resume functionality

Event Handling

The system handles multiple event types:

- Close events- User-initiated dismissal
- Action events- Button clicks and custom actions
- Timeout events- Auto-dismiss completion
- Hover events- Timer pause/resume functionality

DIALOG

The

<ava-dialog>

system provides a comprehensive dialog solution with multiple variants, modal support, and flexible configuration options. Built around a service-based architecture, it offers both predefined dialog types and custom modal dialogs for various user interaction scenarios.

How to use

```
import {  
    AavaDialogService  
} from "@aava/play-core"  
  
// Inject the service  
constructor(  
    private dialogService : AavaDialogService  
) {  
  
    // Show different types of dialogs  
    this.  
        .  
            dialogService
```

```
success
(
{
message
:
'Operation completed!'
}
)
;
this

.
dialogService

.
confirmation
(
{
message
:
'Are you sure?'
}
)
;
```

Basic Usage

Simple dialog implementation with default settings and integrated functionality.

```
<aava-button
label="Show Warning Dialog"
variant="warning"
(userClick)="showWarningDialog()"
></aava-button>
```

```
---  
constructor(private dialogService: DialogService) {}  
  
showWarningDialog() {  
  this.dialogService  
    .warning({  
      title: 'Unsaved Changes',  
      message:  
        'You have unsaved changes that will be lost if you continue. Are you sure you want to proceed?',  
      showProceedButton: true,  
      proceedButtonText: 'Discard Changes',  
    })  
    .then((result) => {  
      console.log('Warning dialog closed:', result);  
      if (result.action === 'proceed') {  
        console.log('User chose to discard changes!');  
      }  
    });  
}  
}
```

Basic Features

- Service-Based: Programmatic dialog creation via DialogService
- Multiple Variants: Success, error, warning, info, confirmation, loading, and custom dialogs
- Modal Support: Full modal dialog capabilities with content projection
- Responsive Design: Adapts to different screen sizes and content
- Accessibility: Built-in accessibility features and keyboard navigation

Dialog Variants

Seven distinct dialog variants for different user interaction scenarios.

[Error extracting code]

Available Variants

- Success- Green styling for successful operations and confirmations
- Error- Red styling for errors and critical issues with retry options
- Warning- Orange/yellow styling for warnings and cautions
- Info- Blue styling for informational messages and tips
- Confirmation- Neutral styling for user confirmations and decisions
- Loading- Animated loading states with progress indicators
- Custom- Fully customizable styling and content

Custom Dialogs

Fully customizable dialogs with custom content and styling options.

[Error extracting code]

Custom Features

- Custom Content: HTML content or component rendering
- Variant Styling: Multiple visual variants (default, success, error, warning, info)
- Flexible Layouts: Custom button configurations and layouts
- Icon Customization: Optional icons with custom colors and sizes
- Bottom Borders: Optional decorative bottom borders

Service Methods

The DialogService provides convenient methods for different dialog types:

Success Dialog

this

```
    .  
    dialogService
```

```
    .  
    success
```

```
(
```

```
{
```

```
title
```

```
:
```

```
"Success!"  
,
```

message

```
:
```

"Operation completed successfully."

```
,
```

size

```
:
```

"lg"

```
,
```

showButtons

```
:
```

true

```
,
```

buttons

```
:
```

[

{

label

```
:
```

"Continue"

```
,
```

variant

```
:
```

"primary"

```
,
```

action

```
:
```

"continue"

```
}
```

```
]
```

```
,
```

```
}
```

```
)
```

```
;
```

Error Dialog

```
this
```

```
dialogService
```

```
error
```

```
(
```

```
{
```

```
title
```

```
:
```

```
"Error!"
```

```
,
```

```
message
```

```
:
```

```
"Something went wrong. Please try again."
```

```
,
```

```
showRetryButton
```

```
:
```

```
true
```

```
,
```

```
retryButtonText
```

```
:
```

```
"Retry"
```

,

size

:

"md"

,

}

)

;

Warning Dialog

this

.

dialogService

.

warning

(

{

title

:

"Warning"

,

message

:

"Please review the information carefully."

,

showProceedButton

:

true

,

proceedButtonText

```
:  
"Proceed"  
,  
showCancelButton  
:  
true  
,  
cancelButtonText  
:  

```

Confirmation Dialog

```
this
```

```
.
```

```
dialogService
```

```
.
```

```
confirmation
```

```
(
```

```
{
```

```
title
```

```
:
```

```
"Delete Item"
```

```
,
```

```
message
```

```
:
```

```
"Are you sure you want to delete this item?"
```

```
,
```

```
confirmButtonText
```

```
:
```

```
"Delete"
```

```
,
```

```
cancelButtonText
```

```
:
```

```
"Cancel"
```

```
,
```

```
confirmButtonVariant
```

```
:
```

```
"danger"
```

```
,
```

```
destructive
```

```
:
```

```
true
```

```
,
```

```
}
```

```
)
```

```
;
```

Loading Dialog

```
this
```

```
.
```

```
dialogService
```

```
.
```

```
loading
```

```
(
```

```
{
```

```
title
```

```
:  
"Processing"  
,  
message  
:  
"Please wait while we process your request."  
,  
showProgress  
:  

```

Custom Dialog

this

```
dialogService

.

custom

(
{

title

:

"Custom Dialog"

,

message

:

"This is a custom dialog with unique content."

,

variant

:

"success"

,

customContent

:

"<div>Custom HTML content here</div>"

,

buttons

:

[

{

label

:

"OK"

,
```

```
variant
:
"primary"
,
action
:
"ok"
}

,
{
label
:
"Cancel"
,
variant
:
"secondary"
,
action
:
"cancel"
}

,
]
,
}
)
;
;
```

Modal Dialog

```
this

.

dialogService

.

openModal
(
MyCustomComponent

,
{
width
:
"600px"

,
maxWidth
:
"90vw"

,
showCloseButton
:
true

,
}

,
{
data
:
{
id
:

```

```
123
,
name
:
"Example"
}
,
}
)
;
```

API Reference

DialogService Methods

[TABLE]

Method | Parameters | Return Type | Description

success() | SuccessDialogConfig? | Promise<DialogResult> | Show success dialog with green styling

error() | ErrorDialogConfig? | Promise<DialogResult> | Show error dialog with red styling

warning() | WarningDialogConfig? | Promise<DialogResult> | Show warning dialog with orange styling

info() | InfoDialogConfig? | Promise<DialogResult> | Show info dialog with blue styling

confirmation() | ConfirmationDialogConfig? | Promise<DialogResult> | Show confirmation dialog for user decisions

loading() | LoadingDialogConfig? | Promise<DialogResult> | Show loading dialog with progress indicators

custom() | CustomDialogConfig? | Promise<DialogResult> | Show custom dialog with full customization

feedback() | CustomDialogConfig? | Promise<DialogResult> | Show feedback dialog for user input

open() | component, data? | Promise<any> | Open custom component in dialog container

openModal() | component, config?, data? | Promise<any> | Open custom component in modal dialog

close() | - | void | Close the currently open dialog

[/TABLE]

Base Dialog Configuration

[TABLE]

Property	Type	Default	Description
title	string?	"	Dialog header text
message	string?	"	Dialog body text
icon	string?	"	Icon name for the dialog
iconColor	string?	"	Custom icon color
iconSize	number?	"	Custom icon size in pixels
showCloseButton	boolean?	true	Whether to show close button
backdrop	boolean?	true	Whether to show backdrop
width	string?	"	Custom dialog width
height	string?	"	Custom dialog height
data	any?	"	Additional data to pass to dialog

[/TABLE]

Success Dialog Configuration

[TABLE]

Property	Type	Default	Description
buttons	DialogButton[]?	[]	Array of custom buttons
showButtons	boolean?	false	Whether to show custom buttons
bottomBorder	boolean?	true	Whether to show decorative bottom border
size	'lg' 'md' 'sm' 'lg'		Dialog size variant

[/TABLE]

Error Dialog Configuration

[TABLE]

Property	Type	Default	Description
showRetryButton	boolean?	false	Whether to show retry button
retryButtonText	string?	'Retry'	Text for retry button
closeButtonText	string?	'Close'	Text for close button
bottomBorder	boolean?	true	Whether to show decorative bottom border
buttons	ErrorButton[]?	[]	Array of custom error buttons

showButtons | boolean? | false | Whether to show custom buttons
size | 'lg' | 'md' | 'sm' | 'lg' | Dialog size variant
[/TABLE]

Warning Dialog Configuration

[TABLE]

Property | Type | Default | Description
showProceedButton | boolean? | false | Whether to show proceed button
proceedButtonText | string? | 'Proceed' | Text for proceed button
showCancelButton | boolean? | true | Whether to show cancel button
cancelButtonText | string? | 'Cancel' | Text for cancel button
bottomBorder | boolean? | true | Whether to show decorative bottom border
buttons | WarningButton[]? | [] | Array of custom warning buttons
showButtons | boolean? | false | Whether to show custom buttons
size | 'lg' | 'md' | 'sm' | 'lg' | Dialog size variant
[/TABLE]

Info Dialog Configuration

[TABLE]

Property | Type | Default | Description
showOkButton | boolean? | true | Whether to show OK button
okButtonText | string? | 'OK' | Text for OK button
showLearnMoreButton | boolean? | false | Whether to show learn more button
learnMoreButtonText | string? | 'Learn More' | Text for learn more button
bottomBorder | boolean? | true | Whether to show decorative bottom border
buttons | InfoButton[]? | [] | Array of custom info buttons
showButtons | boolean? | false | Whether to show custom buttons
size | 'lg' | 'md' | 'sm' | 'lg' | Dialog size variant
[/TABLE]

Confirmation Dialog Configuration

[TABLE]

Property	Type	Default	Description
confirmButtonText	string?	'Confirm'	Text for confirm button
cancelButtonText	string?	'Cancel'	Text for cancel button
confirmButtonVariant	'primary' 'secondary' 'success' 'warning' 'danger' 'primary'	'primary'	Confirm button styling variant
cancelButtonVariant	'primary' 'secondary' 'success' 'warning' 'danger' 'secondary'	'secondary'	Cancel button styling variant
destructive	boolean?	false	Whether this is a destructive action
bottomBorder	boolean?	false	Whether to show decorative bottom border

[/TABLE]

Loading Dialog Configuration

[TABLE]

Property	Type	Default	Description
progress	number?	0	Progress value (0-100)
showProgress	boolean?	false	Whether to show progress bar
showCancelButton	boolean?	false	Whether to show cancel button
cancelButtonText	string?	'Cancel'	Text for cancel button
spinnerColor	string?	"	Custom spinner color
indeterminate	boolean?	true	Whether progress is indeterminate
bottomBorder	boolean?	false	Whether to show decorative bottom border

[/TABLE]

Custom Dialog Configuration

[TABLE]

Property	Type	Default	Description
buttons	DialogButton[]?	[]	Array of custom buttons
variant	'default' 'success' 'error' 'warning' 'info' 'default'	'default'	Visual variant for styling
customContent	string?	"	Custom HTML content
showIcon	boolean?	true	Whether to show icon

showTitle | boolean? | true | Whether to show title
showMessage | boolean? | true | Whether to show message
bottomBorder | boolean? | false | Whether to show decorative bottom border
label | string? | " | Custom label text
confirmButtonText | string? | " | Text for confirm button
cancelButtonText | string? | " | Text for cancel button
destructive | boolean? | false | Whether this is a destructive action
[/TABLE]

Modal Dialog Configuration

[TABLE]

Property | Type | Default | Description
maxWidth | string? | " | Maximum width constraint
maxHeight | string? | " | Maximum height constraint
showCloseButton | boolean? | true | Whether to show close button
[/TABLE]

Dialog Button Interface

[TABLE]

Property | Type | Default | Description
label | string | - | Button display text
variant | 'primary' | 'secondary' | 'success' | 'warning' | 'danger' | 'primary' | Button styling variant
action | string? | " | Action identifier for button click
disabled | boolean? | false | Whether button is disabled
[/TABLE]

Dialog Result Interface

[TABLE]

Property | Type | Description
action | string? | Action that triggered dialog closure
data | any? | Additional data from dialog interaction

confirmed | boolean? | Whether user confirmed the action

[/TABLE]

Best Practices

Design Guidelines

- Clear Purpose: Make dialog purpose obvious through title and content
- Consistent Styling: Use consistent visual hierarchy across dialog types
- Appropriate Sizing: Choose dialog sizes that fit content without overwhelming
- Button Placement: Follow standard button placement conventions
- Visual Feedback: Provide clear visual feedback for user actions

Accessibility

- Keyboard Navigation: Ensure full keyboard accessibility (Tab, Escape, Enter)
- Screen Reader Support: Provide descriptive titles and content
- Focus Management: Trap focus within dialog and restore on close
- ARIA Attributes: Use appropriate ARIA roles and labels
- High Contrast: Maintain sufficient contrast for all text and elements

Performance

- Lazy Loading: Load dialog content only when needed
- Memory Management: Properly clean up dialog instances
- Change Detection: Use OnPush strategy for optimal performance
- Event Handling: Clean up event listeners and subscriptions
- Bundle Optimization: Import only needed dialog types

User Experience

- Clear Actions: Make button actions and consequences clear
- Progressive Disclosure: Use dialogs for focused, single-purpose interactions
- Error Handling: Provide clear error messages and recovery options
- Loading States: Show appropriate loading indicators for long operations
- Responsive Design: Ensure dialogs work well on all screen sizes

Implementation Considerations

- Service Injection: Inject DialogService where dialogs are needed
- Promise Handling: Use async/await or .then() for dialog results
- Error Boundaries: Handle dialog errors gracefully
- State Management: Integrate with application state management
- Testing: Test dialog interactions and accessibility features

Technical Notes

Component Architecture

The dialog system uses a service-based architecture:

- DialogService manages all dialog operations and lifecycle
- DialogContainerComponent provides the base container and backdrop
- Individual Dialog Components handle specific dialog types and styling
- ModalComponent provides advanced modal capabilities with content projection

Dialog Lifecycle

The dialog system manages complete lifecycle:

- Creation: Dynamic component creation via service methods
- Rendering: Automatic DOM insertion and view attachment
- Interaction: Event handling and user input processing
- Cleanup: Proper cleanup of views, components, and event listeners

Content Projection

Modal dialogs support advanced content projection:

- Header Content: Use [dialog-header] attribute for header sections
- Body Content: Use [dialog-body] attribute for main content
- Footer Content: Use [dialog-footer] attribute for action areas
- Dynamic Content: Any component can be rendered inside dialogs

Event Handling

The system handles multiple event types:

- Close Events: User-initiated closure via close button or backdrop

- Button Events: Custom button click handling with action identification
- Result Events: Promise resolution with user action results
- Cleanup Events: Proper cleanup of resources and event listeners

CSS Integration

The component integrates with the design system:

- CSS Variables: Uses semantic CSS variables for theming
- Responsive Design: Adapts to different screen sizes
- State Management: Handles various dialog states
- Accessibility: Maintains proper contrast and focus indicators

POPOVER

The

```
<ava-popover>
```

component provides a flexible overlay system for displaying contextual information, help text, and guided tour content. Built with advanced positioning logic and multiple navigation variants, it's perfect for creating interactive user experiences.

How to use

```
import  
{  
  AavaPopOverComponent  
}  
from  
"@aava/play-core"  
;
```

Basic Usage

Simple popover implementation with default settings and basic content display.

```
<aava-button  
  avaPopover  
    [avaPopoverData]="samplePopoverData"  
    [avaPopoverPosition]="'top'"  
    [avaPopoverArrow]="'center'"  
    [label]="'Show Popover'"  
    [variant]="'primary'"  
    [outlined]="'true'"  
    pressedEffect="ripple"  
></aava-button>  
<aava-button  
  avaPopover  
    [avaPopoverData]="learnMorePopoverData"
```

```
[avaPopoverPosition]="'bottom"
[avaPopoverArrow]="'center"
[avaPopoverShowLearnMore]="'true"
[!label]="'Show with Learn More"
[variant]="'secondary"
[outlined]="'true"
pressedEffect="ripple"
></aava-button>
<aava-button
avaPopover
[avaPopoverData]="multiStepPopoverData"
[avaPopoverPosition]="'right"
[avaPopoverArrow]="'center"
[avaPopoverShowButtons]="'true"
[avaPopoverShowPagination]="'true"
[!label]="'Show Multi-step"
[variant]="'success"
[outlined]="'true"
pressedEffect="ripple"
></aava-button>
---
samplePopoverData: PopOverData[] = [
{
  header: 'Welcome!',
  description: 'This is a simple popover with basic content.',
},
];
learnMorePopoverData: PopOverData[] = [
{
```

```
header: 'Learn More',
description:
'This popover includes a learn more link for additional information.',
learnMoreUrl: 'https://example.com',
},
];
multiStepPopoverData: PopOverData[] = [
{
header: 'Step 1',
description: 'This is the first step of the multi-step popover.',
},
{
header: 'Step 2',
description: 'This is the second step with additional information.',
},
{
header: 'Step 3',
description: 'This is the final step with completion details.',
},
];
```

Basic Features

- Flexible Positioning: Automatic positioning with arrow alignment
- Content Management: Support for header, description, and optional learn more links
- Responsive Design: Adapts to different screen sizes and content lengths
- Accessibility: Built-in accessibility features and keyboard navigation
- Animation: Smooth entrance animations with stretch effects

Positioning Options

Four positioning variants with intelligent arrow placement and automatic boundary detection.

[Error extracting code]

Available Positions

- Top- Appears above the target element with downward arrow
- Bottom- Appears below the target element with upward arrow
- Left- Appears to the left of the target element with rightward arrow
- Right- Appears to the right of the target element with leftward arrow

Arrow Alignment

- Start- Arrow positioned at the beginning of the popover
- Center- Arrow centered on the popover (default)
- End- Arrow positioned at the end of the popover

API Reference

Inputs

[TABLE]

Property | Type | Default | Description

config | PopOverConfig | - | Configuration object for popover behavior

data | PopOverData[] | [] | Array of content items to display

[/TABLE]

PopOverConfig Interface

[TABLE]

Property | Type | Default | Description

arrow | 'start' | 'center' | 'end' | null | null | Arrow alignment position

position | 'top' | 'bottom' | 'left' | 'right' | 'top' | Popover position relative to target

left | number | 0 | Target element left position

top | number | 0 | Target element top position

bottom | number | 0 | Target element bottom position

width | number | 0 | Target element width

height | number | 0 | Target element height

showButtons | boolean? | false | Whether to show navigation buttons

showPagination | boolean? | false | Whether to show page counter
showIcon | boolean? | false | Whether to show icon navigation
showSkip | boolean? | false | Whether to show skip button
showLearnMore | boolean? | false | Whether to show learn more button
[/TABLE]

PopOverData Interface

[TABLE]

Property | Type | Default | Description
header | string | - | Header text for the popover
description | string | - | Description text for the popover
learnMoreUrl | string? | - | Optional URL for learn more functionality

[/TABLE]

Properties

[TABLE]

Property | Type | Description
visible | boolean | Whether the popover is currently visible
currentIndex | number | Current page index for multi-page content
currentData | PopOverData | null | Current content data being displayed
currentPage | string | Current page display string (e.g., "2/5")
hasPrevious | boolean | Whether previous page is available
hasNext | boolean | Whether next page is available
[/TABLE]

Methods

[TABLE]

Method | Parameters | Return Type | Description
previous() | event?: any | void | Navigate to previous page
next() | event?: any | void | Navigate to next page
skip() | event?: any | void | Close popover and skip content

learnMore() | event?: any | void | Open learn more URL in new tab
updatePosition() | - | void | Update popover position and arrow placement
setArrowPosition() | - | void | Set CSS custom properties for arrow positioning
[/TABLE]

CSS Custom Properties

The component uses CSS custom properties for theming and styling:

[TABLE]

Property | Default | Description

--popover-background | Dynamic | Background color of the popover
--popover-text | Dynamic | Text color of the popover
--popover-font | Dynamic | Font family for the popover
--popover-padding | Dynamic | Internal padding of the popover
--popover-border-radius | Dynamic | Border radius of the popover
--popover-max-width | Dynamic | Maximum width of the popover
--popover-z-index | Dynamic | Z-index for layering
--popover-shadow | Dynamic | Box shadow for the popover
--popover-border-color | Dynamic | Border color of the popover
--popover-content-gap | Dynamic | Gap between content sections
--popover-header-font-family | Dynamic | Font family for the header
--popover-header-font-size | Dynamic | Font size for the header
--popover-header-font-weight | Dynamic | Font weight for the header
--popover-header-color | Dynamic | Color for the header text
--popover-header-line-height | Dynamic | Line height for the header
--popover-description-font-family | Dynamic | Font family for the description
--popover-description-font-size | Dynamic | Font size for the description
--popover-description-font-weight | Dynamic | Font weight for the description
--popover-description-color | Dynamic | Color for the description text
--popover-description-line-height | Dynamic | Line height for the description
--popover-footer-margin-top | Dynamic | Top margin for the footer

--popover-pagination-font-size | Dynamic | Font size for pagination text
--popover-pagination-font-weight | Dynamic | Font weight for pagination text
--popover-pagination-color | Dynamic | Color for pagination text
--popover-pagination-font-family | Dynamic | Font family for pagination text
--popover-pagination-line-height | Dynamic | Line height for pagination text
--popover-button-gap | Dynamic | Gap between navigation buttons
--popover-button-width | Dynamic | Width of navigation buttons
--popover-button-height | Dynamic | Height of navigation buttons
--popover-disabled-opacity | Dynamic | Opacity for disabled button states
--popover-hover-background | Dynamic | Background color for button hover states
--popover-hover-border-radius | Dynamic | Border radius for button hover states
--popover-arrow-border-width | Dynamic | Width of arrow border
--popover-arrow-border-width-inner | Dynamic | Width of inner arrow border
--popover-arrow-border-shadow | Dynamic | Color of arrow border shadow
--popover-arrow-color | Dynamic | Color of arrow fill

[/TABLE]

Best Practices

Content Guidelines

- Keep it concise: Popovers work best with short, focused content
- Use clear headers: Make the purpose of the popover immediately clear
- Provide actionable content: Include learn more links or navigation when appropriate
- Consider context: Position popovers to avoid covering important interface elements

Positioning Guidelines

- Choose appropriate positions: Use top/bottom for horizontal content, left/right for vertical content
- Align arrows thoughtfully: Center arrows for balanced content, start/end for directional emphasis
- Avoid boundaries: Ensure popovers don't extend beyond viewport edges
- Consider scrolling: Position popovers to remain visible during page scroll

Navigation Guidelines

- Match user expectations: Use familiar navigation patterns (prev/next, pagination)
- Provide clear feedback: Show current position and available actions
- Include skip options: Allow users to bypass content when appropriate
- Use consistent styling: Maintain visual consistency across navigation elements

Technical Notes

Component Architecture

The popover component uses a modern Angular architecture with:

- PopOverComponent- Main component with positioning and navigation logic
- IconComponentintegration for icon-based navigation
- ViewChild referencesfor DOM manipulation and positioning
- ChangeDetectionStrategy.OnPushfor performance optimization

Positioning System

The component implements an advanced positioning system:

- Automatic positioningbased on target element coordinates
- Arrow alignmentwith CSS custom properties for precise placement
- Boundary detectionto prevent overflow beyond viewport
- Scroll and resize handlingfor responsive positioning

Animation System

The component provides smooth entrance animations:

- Stretch effectsfor natural expansion from target elements
- Position-based animationswith appropriate transform origins
- CSS keyframesfor consistent and performant animations
- Smooth transitionsfor opacity and transform changes

Event Handling

The component manages multiple event types:

- Navigation eventsfor page changes and content progression
- User interactionfor skip, learn more, and navigation actions
- Window eventsfor scroll and resize handling

- Lifecycle events for proper cleanup and memory management

CSS Architecture

The component uses a comprehensive CSS architecture:

- CSS custom properties for theming and customization
- BEM-like naming for maintainable and scalable styles
- Position-based modifiers for different placement variants
- Responsive design with flexible layouts and sizing

Accessibility Features

The component includes built-in accessibility features:

- Keyboard navigation for all interactive elements
- ARIA attributes for screen reader compatibility
- Focus management for proper tab order and focus indicators
- Semantic markup for clear content structure and meaning

MODAL

The Modal component is a powerful overlay system that extends the dialog service to provide custom modal dialogs with full content projection capabilities. It allows developers to create sophisticated modal interfaces with custom content, layouts, and interactions while maintaining consistent styling and behavior.

The modal system is built on top of the

AavaDialogService

and provides a flexible foundation for creating various types of modal dialogs, from simple confirmations to complex forms and content displays.

How to use

```
import
{
    AavaDialogService
}
from
"@aava/play-core"
;
// Inject the service
constructor
(
    private
        dialogService
    :
    AavaDialogService
)
{
}
// Open a custom modal
this
.
```

```
dialogService

.

openModal
(
  MyCustomModalComponent

,
{
  width
  :

  '600px'

,
  maxWidth
  :

  '90vw'
}

)
;
```

Modal Variants

```
<div class="con">

<div class="col-12 col-sm-auto d-flex justify-content-center">
  <aava-button
    label=" Simple Modal"
    variant="secondary"
    (userClick)="openProperSimpleModal()"

  ></aava-button>

  <aava-button
    label=" Feedback Modal"
    variant="secondary"
    (userClick)="openProperFeedbackModal()"

  ></aava-button>
```

```
></aava-button>

<aava-button
  label=" Scrollable Modal"
  variant="secondary"
  (userClick)="openProperScrollableModal()"

></aava-button>

</div>
</div>
---

constructor(private dialogService: AavaDialogService) {}

openProperSimpleModal() {
  this.dialogService.openModal(PoProperSimpleModalComponent, {
    width: '400px',
  });
}

openProperFeedbackModal() {
  this.dialogService.openModal(PoProperFeedbackModalComponent, {
    width: '400px',
  });
}

openProperScrollableModal() {
  this.dialogService.openModal(PoProperScrollableModalComponent, {
    width: '484px',
  });
}

---

import { Component } from "@angular/core";
import { AavaButtonComponent, AavaDialogService } from "play-comp-library";
```

```
@Component({
  selector: "app-proper-simple-modal",
  standalone: true,
  imports: [AavaButtonComponent],
  template: `
    <div style="padding:24px">
      <div class="ava-modal" style="border-radius:8px;">
        <div
          dialog-header
          style="display: flex;
          align-items: center;
          gap: 12px;
          align-self: stretch;">
        >
        <svg
          xmlns="http://www.w3.org/2000/svg"
          width="24"
          height="24"
          viewBox="0 0 24 24"
          fill="none">
          >
          <path
            d="M22 12C22 17.5228 17.5228 22 12 22M22 12C22 6.47715 17.5228 2 12 2M22 12H2M12
22C6.47715 22 2 17.5228 2 12M12 22C9.43223 19.3038 8 15.7233 8 12C8 8.27674 9.43223 4.69615
12 2M12 22C14.5678 19.3038 16 15.7233 16 12C16 8.27674 14.5678 4.69615 12 2M2 12C2 6.47715
6.47715 2 12 2"
            stroke="black"
            stroke-width="1.5"
            stroke-linecap="round"
            stroke-linejoin="round">
        </path>
      </div>
    </div>
  `
```

```
/>

</svg>

<h4
  style="margin-top:0;
  margin-bottom:0;
  line-height:0;
  color:#3B3F46;
  font-family: Mulish;
  font-size:24px;
  font-style: normal;
  font-weight: 700;
  line-height:28px;">

  >

  Heading

</h4>

</div>

<div dialog-body>

<p
  style="align-self: stretch;
  color: #6B7280;
  font-family: Inter;
  font-size:16px;
  font-style: normal;
  font-weight: 400;
  line-height:20px;
  margin:24px 0px;">

  >

  This agent will be sent back for corrections and modifications.

  Kindly comment what needs to be done.
```

```
</p>
</div>
<div dialog-footer style="display:flex; gap:12px">
  <aava-button
    label="Label"
    variant="secondary"
    size="md"
    (userClick)="onClose()"
    height="52px"
    width="165px"
  ></aava-button>
  <aava-button
    label="Label"
    variant="primary"
    size="md"
    (userClick)="onClose()"
    height="52px"
    width="165px"
  ></aava-button>
</div>
</div>
</div>
` ,
})
export class ProperSimpleModalComponent {
  constructor(private dialogService: AavaDialogService) {}
  onClose() {
    this.dialogService.close();
  }
}
```

```
}

---

import { Component } from "@angular/core";
import {
  AavaButtonComponent,
  AavaDialogService,
  AavaCheckboxComponent,
} from "play-comp-library";
@Component({
  selector: "app-proper-scrollable-modal",
  standalone: true,
  imports: [AavaButtonComponent, AavaCheckboxComponent],
  template: `
    <div style="padding:24px">
      <div
        style="display:flex;
        flex-direction: column;
        gap:24px;">
        >
        <div
          dialog-header
          style="
            color: #3B3F46;
            font-family: Mulish;
            font-size: 24px;
            font-style: normal;
            font-weight: 700;
            line-height: 28px;
            justify-content:flex-start;
          ">
    
```

"

>

<h3 style="margin-bottom:0px;">Heading</h3>

</div>

<div dialog-body>

<div>

<h4>Heading small</h4>

<p>

This agent will be sent back for corrections and modifications.

Kindly comment what needs to be done.

</p>

<h4 style="margin: 0 0 8px 0; font-size: 14px;">Heading small</h4>

<p style="margin: 0 0 16px 0; color: #666; font-size: 14px;">

This agent will be sent back for corrections and modifications.

Kindly comment what needs to be done.

</p>

<h4>Heading small</h4>

<p>

This agent will be sent back for corrections and modifications.

Kindly comment what needs to be done.

</p>

<h4>Heading small</h4>

<p>

This agent will be sent back for corrections and modifications.

Kindly comment what needs to be done.

</p>

<div>

<h4>Heading small</h4>

<p>

This agent will be sent back for corrections and modifications.

Kindly comment what needs to be done.

```
</p>

</div>

<div
style="
display: flex;
height: 24px;
padding: 0 0;
align-items: center;
gap: 8px;
align-self: stretch;">

<aava-checkbox variant="default" size="sm"></aava-checkbox>

<label> Accept Terms and Conditions</label>

</div>

</div>

</div>

<div
dialog-footer
style="
display:flex;
justify-content: flex-end;
flex-direction: row;
gap: 10px;
align-items: flex-end;
align-self: stretch;">

</>

<aava-button
```

```
label="Label"
variant="secondary"
size="md"
(userClick)="onClose()"
></aava-button>

<aava-button
label="Label"
variant="primary"
size="md"
(userClick)="onClose()"
></aava-button>

</div>
</div>
</div>
`,
})

export class ProperScrollableModalComponent {
constructor(private dialogService: AavaDialogService) {}
onClose() {
this.dialogService.close();
}
}

---
import { Component } from "@angular/core";
import {
AavaButtonComponent,
AavaDialogService,
AavalconComponent,
AavaTextboxComponent,
```

```
} from "play-comp-library";

@Component({
  selector: "app-proper-feedback-modal",
  standalone: true,
  imports: [AavaButtonComponent, AavalconComponent, AavaTextboxComponent],
  template: `

    <div style="padding:24px">
      <div dialog-header>
        <h3
          style="color: #3B3F46;
          font-family: Mulish;
          font-size: 24px;
          font-style: normal;
          font-weight: 700;
          line-height:28px;">
          >
          Heading
        </h3>
      </div>
      <div dialog-body>
        <div style="text-align: center; margin:24px 0px">
          <aava-icon
            iconName="circle-check"
            iconSize="70px"
            iconColor="green"
          ></aava-icon>
        </div>
        <p
          style="align-self: stretch;">
    
```

```
color: #6B7280;  
text-align: center;  
font-family: Inter;  
font-size: 16px;  
font-style: normal;  
font-weight: 400;  
line-height: 20px;  
margin-bottom:24px"  
>  
This agent will be sent back for corrections and modifications. Kindly  
comment what needs to be done  
</p>  
<div style="margin-bottom:24px">  
<aava-textbox  
label="Feedback Input"  
variant="default"  
placeholder="Enter text here"  
size="xl"  
></aava-textbox>  
</div>  
</div>  
<div dialog-footer style="display:flex; gap:12px">  
<aava-button  
label="Label"  
variant="secondary"  
size="md"  
(userClick)="onClose()"  
height="52px"  
width="165px"
```

```

></aava-button>

<aava-button
  label="Label"
  variant="primary"
  size="md"
  height="52px"
  (userClick)="onClose()"
  width="165px"
></aava-button>

</div>
</div>
` ,
})

```

```

export class ProperFeedbackModalComponent {
  constructor(private dialogService: AavaDialogService) {}

  onClose() {
    this.dialogService.close();
  }
}

```

Simple Modal Features

- Header Section: Custom header with icon and title
- Body Content: Descriptive text content
- Footer Actions: Primary and secondary action buttons
- Responsive Design: Adapts to different screen sizes
- Accessibility: Proper ARIA attributes and keyboard navigation

Feedback Modal Features

- Visual Feedback: Success icon with color-coded styling
- Input Fields: Text input for user feedback
- Centered Layout: Optimized for form interactions

- Action Buttons: Clear primary and secondary actions
- User Experience: Intuitive feedback collection flow

Scollable Modal Features

- Content Scrolling: Handles overflow content gracefully
- Form Elements: Checkbox and form controls
- Structured Content: Organized sections with headings
- Flexible Height: Adapts to content length
- Footer Positioning: Actions remain accessible during scroll

Features

Content Projection

The modal system uses Angular's content projection to provide flexible content structure:

- [dialog-header]: Header section for titles, icons, and navigation
- [dialog-body]: Main content area for forms, text, or components
- `[dialog-footer]: Footer section for action buttons and controls

Flexible Sizing

- Width Control: Customizable width with responsive constraints
- Height Management: Automatic height adjustment or fixed heights
- Responsive Behavior: Adapts to different screen sizes
- Max Dimensions: Configurable maximum width and height

Service Integration

- DialogService: Centralized modal management
- Component Injection: Dynamic component rendering
- Configuration Options: Flexible modal configuration
- Lifecycle Management: Proper cleanup and resource management

Accessibility

- Keyboard Navigation: Full keyboard support (Tab, Escape, Enter)
- Screen Reader: Proper ARIA attributes and labels

- Focus Management: Focus trapping and restoration
- High Contrast: Maintains accessibility standards

API Reference

AavaDialogService.openModal()

openModal

(

component

:

Type

<

any

>

,

config

?

:

ModalConfig

,

data

?

:

any

)

:

Promise

<

any

>

[TABLE]

Property	Type	Default	Description
width	string	-	Modal width (e.g., '600px')
maxWidth	string	-	Maximum width constraint
height	string	-	Modal height
maxHeight	string	-	Maximum height constraint
showCloseButton	boolean	true	Show close button
backdrop	boolean	true	Show backdrop overlay
closeOnBackdrop	boolean	true	Close on backdrop click

[/TABLE]

Content Projection Attributes

[TABLE]

Attribute	Description
[dialog-header]	Header section for modal title and navigation
[dialog-body]	Main content area for modal content
`[dialog-footer]	Footer section for action buttons

[/TABLE]

Best Practices

Modal Design

- Clear Purpose: Make modal purpose obvious through title and content
- Appropriate Sizing: Choose modal sizes that fit content without overwhelming
- Consistent Layout: Use consistent header, body, and footer structure
- Visual Hierarchy: Maintain clear visual hierarchy within the modal
- Responsive Design: Ensure modals work well on all screen sizes

Content Organization

- Header Content: Include clear titles and optional icons
- Body Structure: Organize content logically with proper spacing
- Footer Actions: Place primary actions on the right, secondary on the left

- Content Length: Keep content concise or implement scrolling for long content
- Form Elements: Use appropriate form controls and validation

User Experience

- Clear Actions: Make button actions and consequences clear
- Escape Options: Always provide a way to close the modal
- Loading States: Show appropriate loading indicators for async operations
- Error Handling: Provide clear error messages and recovery options
- Accessibility: Ensure keyboard navigation and screen reader support

Performance

- Lazy Loading: Load modal content only when needed
- Component Reuse: Reuse modal components when possible
- Memory Management: Properly clean up modal instances
- Change Detection: Use OnPush strategy for optimal performance
- Bundle Optimization: Import only needed modal components

Styling

The modal component uses CSS custom properties for theming:

```
:root
{
  --modal-background: #fff;
  --modal-border: 1px solid #e2e8f0;
  --modal-shadow: 0 0 10px 2px #e2e8f0;
}
```

e2e8f0

```
0
20
px
25
px
-5
px
rgba
(
0
,
0
,
0
,
0.1
)
;
--modal-radius
:
8
px
;
--modal-padding
:
24
px
;
--modal-header-color
```

3b3f46

--modal-body-color

6b7280

--modal-footer-border

f1f5f9

;

}

CARD

The AAVA Play card system provides a flexible and modular approach to creating content containers. It consists of a main card container (

aava-default-card

) and dedicated subcomponents for header (

aava-card-header

), content (

aava-card-content

), footer (

aava-card-footer

), and actions (

aava-card-actions

). This modular design allows for consistent, well-structured content layouts across your application.

How to use

import

{

AavaDefaultCardComponent

,

AavaCardHeaderComponent

,

AavaCardContentComponent

,

AavaCardFooterComponent

,

AavaTagComponent

,

}

from

"@aava/play-core"

;

Basic Usage

A simple card with basic content structure.

```
<aava-default-card>

<aava-card-header>

<div class="header-with-badge">

<h4>Article Title</h4>

<aava-tag [pill]="true" label="New" color="primary"></aava-tag>

</div>
```

```
</aava-card-header>
```

```
<aava-card-content>
```

```
<p>
```

A complete card example with header, content, and footer sections. Perfect for articles, blog posts, or detailed information displays.

```
</p>
```

```
</aava-card-content>
```

```
<aava-card-footer>
```

```
<div class="footer-info">
```

```
<div class="meta-info">
```

```
<aava-icon
```

```
iconName="calendar-days"
```

```
[iconSize]="16"
```

```
iconColor="#6c757d"
```

```
></aava-icon>
```

```
<span>Dec 15, 2024</span>
```

```
</div>
```

```
<div class="meta-info">
```

```
<aava-icon
```

```
iconName="user"
```

```
[iconSize="16"
iconColor="#6c757d"
>/>aava-icon>
<span>John Doe</span>
</div>
</div>
</aava-card-footer>
</aava-default-card>
```

With Header

Card with a header section for titles, navigation, or metadata.

```
<aava-default-card>
<aava-card-header>
<div class="header-with-icon">
<aava-icon
iconName="shield-check"
[iconSize="24"
iconColor="#28a745"
>/>aava-icon>
<h4>Security Settings</h4>
</div>
</aava-card-header>
<aava-card-content>
<p>
Icons in headers help convey meaning quickly and add visual interest. This
card shows security-related settings.
</p>
<ul class="settings-list">
<li>Two-factor authentication enabled</li>
```

```
<li>Password strength: Strong</li>
<li>Last login: 2 hours ago</li>
</ul>
</aava-card-content>
</aava-default-card>
```

With Actions

Card with action buttons using the dedicated actions component.

```
<aava-default-card>
<aava-card-header>
<h4>Account Settings</h4>
</aava-card-header>
<aava-card-content>
<p>
  Manage your account preferences, security settings, and personal
  information.
</p>
<ul class="settings-list">
<li>Profile Information</li>
<li>Security & Privacy</li>
<li>Notification Preferences</li>
</ul>
</aava-card-content>
<aava-card-footer>
<div class="primary-actions">
<aava-button
  label="Save Changes"
  variant="primary"
  size="sm"
  (click)="onPrimaryAction('Save Changes')">
```

```
>  
</aava-button>  
</div>  
</aava-card-footer>  
</aava-default-card>  
---  
onPrimaryAction(action: string) {  
  console.log(`Primary action: ${action}`);  
}  
}
```

Example Cards

```
<div class="grid-container">  
  <div class="test-group">  
    <aava-image-card  
      [data]="verticalWithActions"  
      (buttonClicked)="onButtonClick($event)"  
      (cardClick)="onCardClick()">  
    </aava-image-card>  
  </div>  
  <div class="test-group">  
    <aava-image-card  
      [data]="verticalWithoutActions1"  
      (cardClick)="onCardClick()">  
    </aava-image-card>  
  </div>  
  <div class="test-group">  
    <aava-image-card  
      [data]="verticalWithoutActions3"  
      (cardClick)="onCardClick()">
```

```
></aava-image-card>
</div>
<div class="test-group stack">
<aava-image-card
  [data]="horizontalWithoutActions4"
  (cardClick)="onCardClick()"
></aava-image-card>
<aava-image-card
  [data]="horizontalWithoutActions2"
  (cardClick)="onCardClick()"
></aava-image-card>
</div>
</div>
<!-- Horizontal Cards Row -->
<div class="grid-container horizontal-row">
<div class="test-group">
<aava-image-card
  [data]="horizontalWithActions"
  (buttonClicked)="onButtonClick($event)"
  (cardClick)="onCardClick()"
></aava-image-card>
</div>
<div class="test-group"></div>
</div>
---
onButtonClick(event: { action: string | undefined, button: any }) {
  alert(`Button clicked! Text: "${event.button.text}" | Action: ${event.action}`);
  console.log('Button click event:', event);
}
```

```
onCardClick() {  
  console.log('Card clicked!');  
}  
  
// Card 1: Vertical With Actions  
  
verticalWithActions: ImageCardData = {  
  variant: 'withActions',  
  title: 'Gen AI 101 Learning Path',  
  image: '/assets/card-1-v.svg',  
  description: 'Lorem ipsum dolor sit amet, consectetur adipiscing elit. Phasellus vulputate, odio no...',  
  divider: {  
    variant: 'solid',  
    color: '#E5E7EB'  
  },  
  avatar: {  
    url: 'https://images.unsplash.com/photo-1472099645785-5658abf4ff4e?ixlib=rb-1.2.1&ixid=eyJhcHBfaWQiOjEyMDd9&auto=format&fit=facearea&facepad=2&w=256&h=256&q=80',  
    size: 'extra-small',  
    text: 'Jhone Doe'  
  },  
  subText: {  
    icon: {  
      name: 'eye',  
      size: 16,  
      color: '#BBBEC5'  
    },  
    text: '440'  
  },  
  buttons: [  
  ]
```

```
text: 'Button Text',
action: 'start-learning',
variant: 'secondary',
size: 'md',
width: '162px',
height: '44px'

},
{
text: 'Button Text',
action: 'view-details',
variant: 'primary',
size: 'md',
width: '162px',
height: '44px'

}
],
layout: {
orientation: 'vertical',
imageGrid: 'top',
infoGrid: 'bottom'
}
};

// Card 2: Horizontal With Actions

horizontalWithActions: ImageCardData = {

variant: 'withActions',
title: 'Gen AI 101 Learning Path',
image: '/assets/card-1-h.svg',
description: 'Lorem ipsum dolor sit amet, consectetur adipiscing elit. Phasellus vulputate, odio...',
divider: {
```

```
variant: 'solid',
color: '#E5E7EB'
},
avatar: {
url: 'https://images.unsplash.com/photo-1472099645785-5658abf4ff4e?ixlib=rb-1.2.1&ixid=:eyJhcHBfaWQiOjEyMDd9&auto=format&fit=facearea&facepad=2&w=256&h=256&q=80',
size: 'extra-small',
text: 'Jhone Doe'
},
subText: {
icon: {
name: 'eye',
size: 16,
color: '#BBBEC5'
},
text: '440'
},
buttons: [
{
text: 'Button Text',
action: 'start-learning',
variant: 'secondary',
size: 'md',
width: '139px',
height: '44px'
},
{
text: 'Button Text',
action: 'view-details',

```

```
variant: 'primary',
size: 'md',
width: '139px',
height: '44px'
},
],
layout: {
orientation: 'horizontal',
imageGrid: 'left',
infoGrid: 'right'
}
};

// Card 3: Vertical Without Actions (Card 1)
verticalWithoutActions1: ImageCardData = {
variant: 'withoutActions',
title: 'Generate UI Design',
image: '/assets/card-1-v.svg',
description: 'Lorem ipsum dolor sit amet, consectetur adipiscing elit. Phasellus vulputate, odio...',
tags: [
{
label: 'Tag',
size: 'xs',
color: 'primary',
shape: 'square'
}
],
subDescription: {
left: 'Text Casus Generibus(15)',
right: '15th May, 18:00'
```

```
},  
layout: {  
  orientation: 'vertical',  
  imageGrid: 'bottom',  
  infoGrid: 'top'  
}  
};  
  
// Card 4: Horizontal Without Actions (Card 2)  
  
horizontalWithoutActions2: ImageCardData = {  
  variant: 'withoutActions',  
  title: 'Generate UI Design',  
  image: '/assets/card-1-h.svg',  
  description: 'Lorem ipsum dolor sit amet, consectetur adipiscing elit. Phasellus vulputate, odio...',  
  tags: [  
    {  
      label: 'Tag',  
      size: 'xs',  
      color: 'primary',  
      shape: 'square'  
    }  
  ],  
  subDescription: {  
    left: 'Text Casus Generibus(15)',  
    right: '15th May, 18:00'  
  },  
  layout: {  
    orientation: 'horizontal',  
    imageGrid: 'left',  
    infoGrid: 'right'
```

```
}

};

// Card 5: Vertical Without Actions (Card 3)

verticalWithoutActions3: ImageCardData = {

variant: 'withoutActions',

title: 'Generate UI Design',

image: '/assets/card-1-v.svg',

description: 'Lorem ipsum dolor sit amet, consectetur adipiscing elit. Phasellus vulputate, odio...',

tags: [

{

label: 'Tag',

size: 'xs',

color: 'primary',

shape: 'square'

}

],

subDescription: {

left: 'Text Casus Generibus(15)',

right: '15th May, 18:00'

},

layout: {

orientation: 'vertical',

imageGrid: 'top',

infoGrid: 'bottom'

}

};

// Card 6: Horizontal Without Actions (Card 4)

horizontalWithoutActions4: ImageCardData = {

variant: 'withoutActions',
```

```

title: 'Generate UI Design',
image: '/assets/card-1-h.svg',
description: 'Lorem ipsum dolor sit amet, consectetur adipiscing elit. Phasellus vulputate, odio...',
tags: [
{
label: 'Tag',
size: 'xs',
color: 'primary',
shape: 'square'
},
],
subDescription: {
left: 'Text Casus Generibus(15)',
right: '15th May, 18:00'
},
layout: {
orientation: 'horizontal',
imageGrid: 'right',
infoGrid: 'left'
}
};

```

Features

Modular Components

- Default Card(aava-default-card): Main card container with proper styling and accessibility
- Card Header(aava-card-header): Dedicated header component with semantic HTML
- Card Content(aava-card-content): Main content area component
- Card Footer(aava-card-footer): Footer section component
- Card Actions(aava-card-actions): Action buttons container component

Accessibility

- Semantic Structure: Proper ARIA roles and landmarks
- Keyboard Navigation: Full keyboard accessibility
- Screen Reader Support: Clear content structure for assistive technologies

Responsive Design

- Flexible Width: Adapts to container width
- Mobile Friendly: Optimized for all screen sizes
- Consistent Spacing: Maintains proper spacing across devices

API Reference

DefaultCardComponent

The main card container component that provides the base card structure.

Selector:

aava-default-card

[TABLE]

Property | Type | Default | Description

No inputs | - | - | Uses content projection for all content

[/TABLE]

Methods:

None

Events:

None

CardHeaderComponent

Header section component with semantic HTML structure and typography styling.

Selector:

aava-card-header

[TABLE]

Property | Type | Default | Description

No inputs | - | - | Uses content projection for header content

[/TABLE]

Methods:

None

Events:

None

CardContentComponent

Main content area component for displaying primary card content.

Selector:

aava-card-content

[TABLE]

Property | Type | Default | Description

No inputs | - | - | Uses content projection for main content

[/TABLE]

Methods:

None

Events:

None

CardFooterComponent

Footer section component for additional information or metadata.

Selector:

aava-card-footer

[TABLE]

Property | Type | Default | Description

No inputs | - | - | Uses content projection for footer content

[/TABLE]

Methods:

None

Events:

None

CardActionsComponent

Action buttons container component for user interactions.

Selector:

aava-card-actions

[TABLE]

Property | Type | Default | Description

No inputs | - | - | Uses content projection for action buttons

[/TABLE]

Methods:

None

Events:

None

CSS Custom Properties

The card components use the following CSS custom properties for styling:

[TABLE]

Property | Description | Default Value

--card-default-border | Border color for the default card | Defined in theme

--tooltip-shadow | Box shadow for the card | Defined in theme

--card-heading1-font | Font family for h1 elements in header | Defined in theme

--card-heading1-font-weight | Font weight for h1 elements in header | Defined in theme

[/TABLE]

Component Styling

Default Card:

- Border radius: 24px
- Border: 1px solid with--card-default-border
- Background: Linear gradient from white to light gray
- Box shadow: Uses--tooltip-shadow
- Padding: 24px

- Width: 100%

Card Header:

- Uses semantic <header> element with role="banner"
- Typography styling for h1 elements
- Custom font family and weight support

Card Content:

- Clean content area with no additional styling
- Uses content projection for flexible content

Card Footer:

- Clean footer area with no additional styling
- Uses content projection for flexible content

Card Actions:

- Clean actions area with no additional styling
- Uses content projection for flexible content

Best Practices

Design Guidelines

- Content Hierarchy: Use headers for titles, content for main information, and footers for actions or metadata
- Consistent Spacing: Maintain consistent padding and margins across all cards
- Visual Balance: Ensure content is well-distributed within the card
- Clear Boundaries: Use borders or shadows to clearly define card boundaries

Component Usage

- Always use `aava-default-card` as the main container
- Use dedicated subcomponents for each section (header, content, footer, actions)
- Maintain semantic structure with proper heading hierarchy
- Keep components focused on their specific responsibilities

Accessibility

- Semantic Structure: Use proper heading hierarchy within card headers

- Focus Management: Ensure logical tab order through card content
- Screen Reader Support: Provide descriptive content for assistive technologies
- Color Contrast: Maintain sufficient contrast for all text content

Performance

- Modular Design: Use only the components you need
- Change Detection: Leverage OnPush strategy for better performance
- Lazy Loading: Consider lazy loading for cards with heavy content
- Memory Management: Clean up event listeners and subscriptions

GRID

The Grid system is a comprehensive CSS-based layout framework that provides responsive, mobile-first grid classes for building flexible and adaptive layouts. Based on Bootstrap's flexbox grid system, it includes containers, rows, columns, and extensive flexbox utilities with breakpoint-specific behavior.

How to use

The Grid system is implemented as CSS classes that can be applied directly to HTML elements. No JavaScript or Angular components are required.

```
<
div
class
=
"
container
"
>
<
div
class
=
"
row
"
>
<
div
class
=
"
col-md-6
```

```
">  
Column 1  
</  
div  
>  
<  
div  
class  
=  
"  
col-md-6  
"  
>  
Column 2  
</  
div  
>  
</  
div  
>  
</  
div  
>
```

Basic Grid

Simple grid layout with equal-width columns.

```
<div class="grid-basic-demo">  
<h3>Basic Grid Layout</h3>  
<div class="demo-section">
```

```
<h4>Equal Width Columns</h4>

<p>
Basic grid with equal-width columns using the <code>.col</code> class.
</p>

<div class="container">
<div class="row">
<div class="col">
<div class="grid-item">
<h5>Column 1</h5>
<p>This column automatically sizes to fill available space.</p>
</div>
</div>
<div class="col">
<div class="grid-item">
<h5>Column 2</h5>
<p>This column also automatically sizes to fill available space.</p>
</div>
</div>
<div class="col">
<div class="grid-item">
<h5>Column 3</h5>
<p>All columns in this row have equal width.</p>
</div>
</div>
</div>
</div>
</div>
<div class="demo-section">
```

```
<h4>Fixed Width Columns</h4>

<p>Grid with specific column widths using numbered classes.</p>

<div class="container">
  <div class="row">
    <div class="col-8">
      <div class="grid-item primary">
        <h5>Main Content (8/12)</h5>
        <p>This column takes up 8 out of 12 grid columns (66.67% width).</p>
      </div>
    </div>
    <div class="col-4">
      <div class="grid-item secondary">
        <h5>Sidebar (4/12)</h5>
        <p>This column takes up 4 out of 12 grid columns (33.33% width).</p>
      </div>
    </div>
    </div>
    </div>
    <div class="demo-section">
      <h4>Mixed Column Types</h4>
      <p>Combining fixed-width and auto-width columns.</p>
      <div class="container">
        <div class="row">
          <div class="col-6">
            <div class="grid-item">
              <h5>Fixed Width (6/12)</h5>
              <p>This column has a fixed width of 50%.</p>
            </div>
          </div>
        </div>
      </div>
    </div>
```

```
</div>

<div class="col">
  <div class="grid-item">
    <h5>Auto Width</h5>
    <p>This column automatically sizes to fill remaining space.</p>
  </div>
</div>

<div class="col-auto">
  <div class="grid-item compact">
    <h5>Auto Size</h5>
    <p>Fits content.</p>
  </div>
</div>
</div>
</div>
</div>

<div class="demo-section">
  <h4>Container Types</h4>
  <p>Different container types for different layout needs.</p>
  <div class="container">
    <div class="row">
      <div class="col-12">
        <div class="grid-item">
          <h5>Fixed Container</h5>
          <p>
            This uses <code>.container</code> with responsive max-widths and
            centered content.
          </p>
        </div>
      </div>
    </div>
  </div>
```

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

<div class="container-fluid">
<div class="row">
<div class="col-12">
<div class="grid-item fluid">
<h5>Fluid Container</h5>
<p>
This uses <code>.container-fluid</code> for full-width layout
spanning the entire viewport.
</p>
</div>
</div>
</div>
</div>
</div>
</div>

<div class="demo-section">
<h4>No Gutters</h4>
<p>Grid without gutters using the <code>.no-gutters</code> class.</p>
<div class="container">
<div class="row no-gutters">
<div class="col-4">
<div class="grid-item no-gutter">
<h5>No Gutter</h5>
<p>This column has no spacing between it and adjacent columns.</p>
</div>
</div>
<div class="col-4">
```

```
<div class="grid-item no-gutter">  
  <h5>No Gutter</h5>  
  <p>Adjacent columns touch each other with no gap.</p>  
</div>  
</div>  
  
<div class="col-4">  
  <div class="grid-item no-gutter">  
    <h5>No Gutter</h5>  
    <p>Perfect for seamless layouts and full-width designs.</p>  
</div>  
</div>  
</div>  
</div>  
</div>  
  
<div class="demo-info">  
  <h4>Basic Grid Features</h4>  
  <ul>  
    <li>  
      <strong>Container System:</strong> Fixed-width and fluid containers  
    </li>  
    <li>  
      <strong>Row Wrapper:</strong> Flexbox rows with automatic gutter  
      management  
    </li>  
    <li>  
      <strong>Column Classes:</strong> Equal-width, fixed-width, and  
      auto-width options  
    </li>  
    <li>
```

```
<strong>No Gutters:</strong> Option to remove spacing between columns
</li>
<li>
<strong>Responsive Foundation:</strong> Mobile-first approach with
responsive breakpoints
</li>
</ul>
</div>
</div>

---
import { Component } from "@angular/core";
@Component({
  selector: "app-grid-basic",
  standalone: true,
  template: `
    <div class="grid-basic-demo">
      <h3>Basic Grid Layout</h3>
      <div class="demo-section">
        <h4>Equal Width Columns</h4>
        <p>
          Basic grid with equal-width columns using the <code>.col</code> class.
        </p>
        <div class="container">
          <div class="row">
            <div class="col">
              <div class="grid-item">
                <h5>Column 1</h5>
                <p>This column automatically sizes to fill available space.</p>
              </div>
            </div>
          </div>
        </div>
      </div>
    </div>
  `})
export class GridBasicComponent {}
```

```
</div>

<div class="col">
  <div class="grid-item">
    <h5>Column 2</h5>
    <p>
      This column also automatically sizes to fill available space.
    </p>
  </div>
</div>

<div class="col">
  <div class="grid-item">
    <h5>Column 3</h5>
    <p>All columns in this row have equal width.</p>
  </div>
</div>
<div>
</div>
<div class="demo-section">
  <h4>Fixed Width Columns</h4>
  <p>Grid with specific column widths using numbered classes.</p>
  <div class="container">
    <div class="row">
      <div class="col-8">
        <div class="grid-item primary">
          <h5>Main Content (8/12)</h5>
          <p>
            This column takes up 8 out of 12 grid columns (66.67% width).
          </p>
        </div>
      </div>
    </div>
  </div>
</div>
```

```
</div>
</div>
<div class="col-4">
<div class="grid-item secondary">
<h5>Sidebar (4/12)</h5>
<p>
This column takes up 4 out of 12 grid columns (33.33% width).
</p>
</div>
</div>
</div>
</div>
</div>
</div>
<div class="demo-section">
<h4>Mixed Column Types</h4>
<p>Combining fixed-width and auto-width columns.</p>
<div class="container">
<div class="row">
<div class="col-6">
<div class="grid-item">
<h5>Fixed Width (6/12)</h5>
<p>This column has a fixed width of 50%.</p>
</div>
</div>
<div class="col">
<div class="grid-item">
<h5>Auto Width</h5>
<p>This column automatically sizes to fill remaining space.</p>
</div>
```

```
</div>

<div class="col-auto">
  <div class="grid-item compact">
    <h5>Auto Size</h5>
    <p>Fits content.</p>
  </div>
</div>
</div>
</div>
</div>
</div>

<div class="demo-section">
  <h4>Container Types</h4>
  <p>Different container types for different layout needs.</p>
  <div class="container">
    <div class="row">
      <div class="col-12">
        <div class="grid-item">
          <h5>Fixed Container</h5>
          <p>
            This uses <code>.container</code> with responsive max-widths
            and centered content.
          </p>
        </div>
      </div>
    </div>
    <div class="container-fluid">
      <div class="row">
        <div class="col-12">
```

```
<div class="grid-item fluid">  
  <h5>Fluid Container</h5>  
  <p>  
    This uses <code>.container-fluid</code> for full-width layout  
    spanning the entire viewport.  
  </p>  
  </div>  
  </div>  
  </div>  
  </div>  
  </div>  
  </div>  
  <div class="demo-section">  
    <h4>No Gutters</h4>  
    <p>Grid without gutters using the <code>.no-gutters</code> class.</p>  
    <div class="container">  
      <div class="row no-gutters">  
        <div class="col-4">  
          <div class="grid-item no-gutter">  
            <h5>No Gutter</h5>  
            <p>  
              This column has no spacing between it and adjacent columns.  
            </p>  
          </div>  
        </div>  
        <div class="col-4">  
          <div class="grid-item no-gutter">  
            <h5>No Gutter</h5>  
            <p>Adjacent columns touch each other with no gap.</p>  
          </div>  
        </div>  
      </div>  
    </div>
```

```
</div>

<div class="col-4">
  <div class="grid-item no-gutter">
    <h5>No Gutter</h5>
    <p>Perfect for seamless layouts and full-width designs.</p>
  </div>
</div>
</div>
</div>
</div>
<div class="demo-info">
  <h4>Basic Grid Features</h4>
  <ul>
    <li>
      <strong>Container System:</strong> Fixed-width and fluid containers
    </li>
    <li>
      <strong>Row Wrapper:</strong> Flexbox rows with automatic gutter
      management
    </li>
    <li>
      <strong>Column Classes:</strong> Equal-width, fixed-width, and
      auto-width options
    </li>
    <li>
      <strong>No Gutters:</strong> Option to remove spacing between
      columns
    </li>
    <li>
```

```
<strong>Responsive Foundation:</strong> Mobile-first approach with
responsive breakpoints

</li>

</ul>

</div>

</div>

`,

styles: [


.grid-basic-demo {
padding: 20px;
max-width: 1200px;
margin: 0 auto;
}

.grid-basic-demo h3 {
margin: 0 0 20px 0;
color: #333;
font-size: 24px;
font-weight: 600;
}

.demo-section {
margin-bottom: 40px;
padding: 20px;
border: 1px solid #e5e7eb;
border-radius: 8px;
background: #f9fafb;
}

.demo-section h4 {
margin: 0 0 12px 0;
```

```
color: #333;  
font-size: 18px;  
font-weight: 600;  
}  
  
.demo-section p {  
margin: 0 0 20px 0;  
color: #555;  
line-height: 1.5;  
}  
  
.demo-section code {  
background: #f1f3f4;  
padding: 2px 6px;  
border-radius: 4px;  
font-family: monospace;  
font-size: 14px;  
}  
  
.grid-item {  
background: #fff;  
border: 1px solid #e5e7eb;  
border-radius: 6px;  
padding: 20px;  
height: 100%;  
min-height: 120px;  
display: flex;  
flex-direction: column;  
justify-content: center;  
}  
  
.grid-item h5 {  
margin: 0 0 12px 0;
```

```
color: #333;  
font-size: 16px;  
font-weight: 600;  
}  
.grid-item p {  
margin: 0;  
color: #666;  
line-height: 1.4;  
}  
.grid-item.primary {  
background: #e3f2fd;  
border-color: #2196f3;  
}  
.grid-item.secondary {  
background: #f3e5f5;  
border-color: #9c27b0;  
}  
.grid-item.compact {  
min-height: 80px;  
padding: 15px;  
}  
.grid-item.fluid {  
background: #e8f5e8;  
border-color: #4caf50;  
}  
.grid-item.no-gutter {  
border-radius: 0;  
border-left: none;  
}
```

```
.grid-item.no-gutter:first-child {  
    border-left: 1px solid #e5e7eb;  
    border-top-left-radius: 6px;  
    border-bottom-left-radius: 6px;  
}  
  
.grid-item.no-gutter:last-child {  
    border-top-right-radius: 6px;  
    border-bottom-right-radius: 6px;  
}  
  
.demo-info {  
    margin-top: 40px;  
    padding: 20px;  
    background: #f8f9fa;  
    border: 1px solid #dee2e6;  
    border-radius: 8px;  
}  
  
.demo-info h4 {  
    margin: 0 0 16px 0;  
    color: #333;  
    font-size: 18px;  
    font-weight: 600;  
}  
  
.demo-info ul {  
    margin: 0;  
    padding-left: 20px;  
    color: #555;  
}  
  
.demo-info li {  
    margin-bottom: 8px;
```

```
line-height: 1.4;  
}  
.demo-info strong {  
color: #333;  
}  
,[,  
})  
  
export class GridBasicComponent {  
constructor() {  
console.log("Grid Basic Component initialized");  
}  
}
```

Responsive Grid

Grid layout that adapts to different screen sizes using breakpoint-specific classes.

```
<div class="grid-responsive-demo">  
<h3>Responsive Grid Layout</h3>  
<div class="demo-section">  
<h4>Responsive Column Classes</h4>  
<p>  
Grid columns that adapt to different screen sizes using  
breakpoint-specific classes.  
</p>  
<div class="container">  
<div class="row">  
<div class="col-12 col-sm-6 col-md-4 col-lg-3">  
<div class="grid-item responsive">  
<h5>Responsive Column</h5>  
<p>
```

```
<strong>Mobile:</strong> 12/12 (100%)<br />
<strong>Small:</strong> 6/12 (50%)<br />
<strong>Medium:</strong> 4/12 (33.33%)<br />
<strong>Large:</strong> 3/12 (25%)
</p>
</div>
</div>

<div class="col-12 col-sm-6 col-md-4 col-lg-3">
<div class="grid-item responsive">
<h5>Responsive Column</h5>
<p>
<strong>Mobile:</strong> 12/12 (100%)<br />
<strong>Small:</strong> 6/12 (50%)<br />
<strong>Medium:</strong> 4/12 (33.33%)<br />
<strong>Large:</strong> 3/12 (25%)
</p>
</div>
</div>

<div class="col-12 col-sm-6 col-md-4 col-lg-3">
<div class="grid-item responsive">
<h5>Responsive Column</h5>
<p>
<strong>Mobile:</strong> 12/12 (100%)<br />
<strong>Small:</strong> 6/12 (50%)<br />
<strong>Medium:</strong> 4/12 (33.33%)<br />
<strong>Large:</strong> 3/12 (25%)
</p>
</div>
```

```
</div>

<div class="col-12 col-sm-6 col-md-4 col-lg-3">
  <div class="grid-item responsive">
    <h5>Responsive Column</h5>
    <p>
      <strong>Mobile:</strong> 12/12 (100%)<br />
      <strong>Small:</strong> 6/12 (50%)<br />
      <strong>Medium:</strong> 4/12 (33.33%)<br />
      <strong>Large:</strong> 3/12 (25%)
    </p>
  </div>
</div>
<div>
</div>
<div>
</div>
<div>
</div>
<div class="demo-section">
  <h4>Mobile-First Layout</h4>
  <p>
    Layout that starts with mobile design and progressively enhances for
    larger screens.
  </p>
  <div class="container">
    <div class="row">
      <div class="col-12 col-md-8">
        <div class="grid-item primary">
          <h5>Main Content</h5>
          <p>
            <strong>Mobile:</strong> Full width (12/12)<br />
            <strong>Medium+:</strong> 8/12 width
          </p>
        </div>
      </div>
    </div>
  </div>
</div>
```

```
</p>

<p>
This column takes full width on mobile devices and 8/12 width on
medium screens and larger.
</p>

</div>
</div>

<div class="col-12 col-md-4">
<div class="grid-item secondary">
<h5>Sidebar</h5>
<p>
<strong>Mobile:</strong> Full width (12/12)<br />
<strong>Medium+:</strong> 4/12 width
</p>
<p>
This sidebar appears below the main content on mobile and beside
it on larger screens.
</p>
</div>
</div>
</div>
</div>
</div>
</div>
<div class="demo-section">
<h4>Complex Responsive Layout</h4>
<p>
Advanced responsive layout with different column configurations at each
breakpoint.
</p>
```

```
<div class="container">  
  <div class="row">  
    <div class="col-12 col-sm-6 col-lg-4">  
      <div class="grid-item">  
        <h5>Feature 1</h5>  
        <p>  
          <strong>Mobile:</strong> 12/12 (100%)<br />  
          <strong>Small:</strong> 6/12 (50%)<br />  
          <strong>Large:</strong> 4/12 (33.33%)  
        </p>  
      </div>  
    </div>  
    <div class="col-12 col-sm-6 col-lg-4">  
      <div class="grid-item">  
        <h5>Feature 2</h5>  
        <p>  
          <strong>Mobile:</strong> 12/12 (100%)<br />  
          <strong>Small:</strong> 6/12 (50%)<br />  
          <strong>Large:</strong> 4/12 (33.33%)  
        </p>  
      </div>  
    </div>  
    <div class="col-12 col-sm-12 col-lg-4">  
      <div class="grid-item">  
        <h5>Feature 3</h5>  
        <p>  
          <strong>Mobile:</strong> 12/12 (100%)<br />  
          <strong>Small:</strong> 12/12 (100%)<br />  
          <strong>Large:</strong> 4/12 (33.33%)  
        </p>  
      </div>  
    </div>  
  </div>  
</div>
```

```
</p>
</div>
</div>
</div>
</div>
</div>

<div class="demo-section">
<h4>Responsive Navigation</h4>
<p>Navigation layout that adapts to different screen sizes.</p>
<div class="container">
<div class="row align-items-center">
<div class="col-12 col-md-3">
<div class="grid-item nav-brand">
<h5>Brand Logo</h5>
<p>Always full width on mobile, 3/12 on medium+</p>
</div>
</div>
<div class="col-12 col-md-9">
<div class="grid-item nav-menu">
<h5>Navigation Menu</h5>
<p>Full width on mobile, 9/12 on medium+</p>
<div class="nav-items">
<span class="nav-item">Home</span>
<span class="nav-item">About</span>
<span class="nav-item">Services</span>
<span class="nav-item">Contact</span>
</div>
</div>
</div>
```

```
</div>
</div>
</div>
<div class="demo-section">
  <h4>Responsive Form Layout</h4>
  <p>
    Form fields that stack on mobile and align side-by-side on larger screens.
  </p>
  <div class="container">
    <div class="row">
      <div class="col-12 col-sm-6">
        <div class="grid-item form-field">
          <h5>First Name</h5>
          <p>Full width on mobile, 6/12 on small+</p>
        </div>
      </div>
      <div class="col-12 col-sm-6">
        <div class="grid-item form-field">
          <h5>Last Name</h5>
          <p>Full width on mobile, 6/12 on small+</p>
        </div>
      </div>
      <div class="col-12">
        <div class="grid-item form-field">
          <h5>Email Address</h5>
          <p>Always full width across all screen sizes</p>
        </div>
      </div>
      <div class="col-12 col-md-8">
```

```
<div class="grid-item form-field">  
  <h5>Message</h5>  
  <p>Full width on mobile, 8/12 on medium+</p>  
</div>  
</div>  
  
<div class="col-12 col-md-4">  
  <div class="grid-item form-field">  
    <h5>Submit</h5>  
    <p>Full width on mobile, 4/12 on medium+</p>  
</div>  
</div>  
</div>  
</div>  
</div>  
  
<div class="demo-info">  
  <h4>Responsive Grid Features</h4>  
  <ul>  
    <li>  
      <strong>Mobile-First:</strong> Design for mobile devices first, then  
      enhance for larger screens  
    </li>  
    <li>  
      <strong>Breakpoint Classes:</strong> Use <code>.col-sm-*</code>,  
      <code>.col-md-*</code>, <code>.col-lg-*</code>, <code>.col-xl-*</code>  
    </li>  
    <li>  
      <strong>Progressive Enhancement:</strong> Start with basic layout, add  
      complexity for larger screens  
    </li>  
  </ul>  
</div>
```

```
<li>

<strong>Content Priority:</strong> Ensure important content is visible
on all screen sizes

</li>

<li>

<strong>Touch Optimization:</strong> Layouts are optimized for touch
interaction on mobile

</li>

</ul>

<h4>Breakpoint Reference</h4>

<ul>

<li>

<strong>Extra Small (xs):</strong> 0px and up (default, no prefix)

</li>

<li><strong>Small (sm):</strong> 576px and up</li>

<li><strong>Medium (md):</strong> 768px and up</li>

<li><strong>Large (lg):</strong> 992px and up</li>

<li><strong>Extra Large (xl):</strong> 1200px and up</li>

</ul>

</div>

</div>

---

import { Component } from "@angular/core";

@Component({  
  selector: "app-grid-responsive",  
  standalone: true,  
  template: `  
    <div class="grid-responsive-demo">  
      <h3>Responsive Grid Layout</h3>
```

```
<div class="demo-section">  
  <h4>Responsive Column Classes</h4>  
  <p>  
    Grid columns that adapt to different screen sizes using  
    breakpoint-specific classes.  
  </p>  
  <div class="container">  
    <div class="row">  
      <div class="col-12 col-sm-6 col-md-4 col-lg-3">  
        <div class="grid-item responsive">  
          <h5>Responsive Column</h5>  
          <p>  
            <strong>Mobile:</strong> 12/12 (100%)<br />  
            <strong>Small:</strong> 6/12 (50%)<br />  
            <strong>Medium:</strong> 4/12 (33.33%)<br />  
            <strong>Large:</strong> 3/12 (25%)  
          </p>  
        </div>  
      </div>  
      <div class="col-12 col-sm-6 col-md-4 col-lg-3">  
        <div class="grid-item responsive">  
          <h5>Responsive Column</h5>  
          <p>  
            <strong>Mobile:</strong> 12/12 (100%)<br />  
            <strong>Small:</strong> 6/12 (50%)<br />  
            <strong>Medium:</strong> 4/12 (33.33%)<br />  
            <strong>Large:</strong> 3/12 (25%)  
          </p>  
        </div>  
      </div>  
    </div>  
  </div>
```

```
</div>

<div class="col-12 col-sm-6 col-md-4 col-lg-3">
<div class="grid-item responsive">
<h5>Responsive Column</h5>
<p>
<strong>Mobile:</strong> 12/12 (100%)<br />
<strong>Small:</strong> 6/12 (50%)<br />
<strong>Medium:</strong> 4/12 (33.33%)<br />
<strong>Large:</strong> 3/12 (25%)<br />
</p>
</div>
</div>

<div class="col-12 col-sm-6 col-md-4 col-lg-3">
<div class="grid-item responsive">
<h5>Responsive Column</h5>
<p>
<strong>Mobile:</strong> 12/12 (100%)<br />
<strong>Small:</strong> 6/12 (50%)<br />
<strong>Medium:</strong> 4/12 (33.33%)<br />
<strong>Large:</strong> 3/12 (25%)<br />
</p>
</div>
</div>
</div>
</div>
<div class="demo-section">
<h4>Mobile-First Layout</h4>
<p>
```

Layout that starts with mobile design and progressively enhances for larger screens.

```
</p>

<div class="container">
  <div class="row">
    <div class="col-12 col-md-8">
      <div class="grid-item primary">
        <h5>Main Content</h5>
        <p>
          <strong>Mobile:</strong> Full width (12/12)<br />
          <strong>Medium+:</strong> 8/12 width
        </p>
      </div>
    </div>
```

This column takes full width on mobile devices and 8/12 width on medium screens and larger.

```
</p>
  </div>
  </div>
  <div class="col-12 col-md-4">
    <div class="grid-item secondary">
      <h5>Sidebar</h5>
      <p>
        <strong>Mobile:</strong> Full width (12/12)<br />
        <strong>Medium+:</strong> 4/12 width
      </p>
    </div>
  </div>
```

This sidebar appears below the main content on mobile and beside it on larger screens.

```
</p>
```

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

<div class="demo-section">
  <h4>Complex Responsive Layout</h4>
  <p>
    Advanced responsive layout with different column configurations at
    each breakpoint.
  </p>
  <div class="container">
    <div class="row">
      <div class="col-12 col-sm-6 col-lg-4">
        <div class="grid-item">
          <h5>Feature 1</h5>
          <p>
            <strong>Mobile:</strong> 12/12 (100%)<br />
            <strong>Small:</strong> 6/12 (50%)<br />
            <strong>Large:</strong> 4/12 (33.33%)
          </p>
        </div>
      </div>
      <div class="col-12 col-sm-6 col-lg-4">
        <div class="grid-item">
          <h5>Feature 2</h5>
          <p>
            <strong>Mobile:</strong> 12/12 (100%)<br />
            <strong>Small:</strong> 6/12 (50%)<br />
          </p>
        </div>
      </div>
    </div>
  </div>
```

```
<strong>Large:</strong> 4/12 (33.33%)  
</p>  
</div>  
</div>  
<div class="grid-item">  
<h5>Feature 3</h5>  
<p>  
<strong>Mobile:</strong> 12/12 (100%)<br />  
<strong>Small:</strong> 12/12 (100%)<br />  
<strong>Large:</strong> 4/12 (33.33%)  
</p>  
</div>  
</div>  
</div>  
</div>  
</div>  
</div>  
<h4>Responsive Navigation</h4>  
<p>Navigation layout that adapts to different screen sizes.</p>  
<div class="container">  
<div class="row align-items-center">  
<div class="col-12 col-md-3">  
<div class="grid-item nav-brand">  
<h5>Brand Logo</h5>  
<p>Always full width on mobile, 3/12 on medium+</p>  
</div>  
</div>  
<div class="col-12 col-md-9">
```

```
<div class="grid-item nav-menu">  
  <h5>Navigation Menu</h5>  
  <p>Full width on mobile, 9/12 on medium+</p>  
  <div class="nav-items">  
    <span class="nav-item">Home</span>  
    <span class="nav-item">About</span>  
    <span class="nav-item">Services</span>  
    <span class="nav-item">Contact</span>  
  </div>  
  </div>  
  </div>  
  </div>  
  </div>  
  </div>  
  </div>  
  <div class="demo-section">  
    <h4>Responsive Form Layout</h4>  
    <p>  
      Form fields that stack on mobile and align side-by-side on larger  
      screens.  
    </p>  
    <div class="container">  
      <div class="row">  
        <div class="col-12 col-sm-6">  
          <div class="grid-item form-field">  
            <h5>First Name</h5>  
            <p>Full width on mobile, 6/12 on small+</p>  
          </div>  
        </div>  
        <div class="col-12 col-sm-6">
```

```
<div class="grid-item form-field">  
  <h5>Last Name</h5>  
  <p>Full width on mobile, 6/12 on small+</p>  
  </div>  
</div>  
  
<div class="col-12">  
  <div class="grid-item form-field">  
    <h5>Email Address</h5>  
    <p>Always full width across all screen sizes</p>  
  </div>  
</div>  
  
<div class="col-12 col-md-8">  
  <div class="grid-item form-field">  
    <h5>Message</h5>  
    <p>Full width on mobile, 8/12 on medium+</p>  
  </div>  
</div>  
  
<div class="col-12 col-md-4">  
  <div class="grid-item form-field">  
    <h5>Submit</h5>  
    <p>Full width on mobile, 4/12 on medium+</p>  
  </div>  
</div>  
</div>  
</div>  
  
<div class="demo-info">  
  <h4>Responsive Grid Features</h4>  
  <ul>
```

-

Mobile-First: Design for mobile devices first, then enhance for larger screens

-

Breakpoint Classes: Use `.col-sm-*`, `.col-md-*`, `.col-lg-*`, `.col-xl-*`

-

Progressive Enhancement: Start with basic layout, add complexity for larger screens

-

Content Priority: Ensure important content is visible on all screen sizes

-

Touch Optimization: Layouts are optimized for touch interaction on mobile

<h4>Breakpoint Reference</h4>

-

Extra Small (xs): 0px and up (default, no prefix)

- Small (sm): 576px and up
- Medium (md): 768px and up

```
<li><strong>Large (lg):</strong> 992px and up</li>
<li><strong>Extra Large (xl):</strong> 1200px and up</li>
</ul>
</div>
</div>
` ,
styles: [
` 

.grid-responsive-demo {
padding: 20px;
max-width: 1200px;
margin: 0 auto;
}

.grid-responsive-demo h3 {
margin: 0 0 20px 0;
color: #333;
font-size: 24px;
font-weight: 600;
}

.demo-section {
margin-bottom: 40px;
padding: 20px;
border: 1px solid #e5e7eb;
border-radius: 8px;
background: #f9fafb;
}

.demo-section h4 {
margin: 0 0 12px 0;
color: #333;
```

```
font-size: 18px;  
font-weight: 600;  
}  
.demo-section p {  
margin: 0 0 20px 0;  
color: #555;  
line-height: 1.5;  
}  
.demo-section code {  
background: #f1f3f4;  
padding: 2px 6px;  
border-radius: 4px;  
font-family: monospace;  
font-size: 14px;  
}  
.grid-item {  
background: #fff;  
border: 1px solid #e5e7eb;  
border-radius: 6px;  
padding: 20px;  
height: 100%;  
min-height: 120px;  
display: flex;  
flex-direction: column;  
justify-content: center;  
}  
.grid-item h5 {  
margin: 0 0 12px 0;  
color: #333;
```

```
font-size: 16px;  
font-weight: 600;  
}  
.grid-item p {  
margin: 0;  
color: #666;  
line-height: 1.4;  
}  
.grid-item.responsive {  
background: #fff3cd;  
border-color: #ffc107;  
}  
.grid-item.primary {  
background: #e3f2fd;  
border-color: #2196f3;  
}  
.grid-item.secondary {  
background: #f3e5f5;  
border-color: #9c27b0;  
}  
.grid-item.nav-brand {  
background: #e8f5e8;  
border-color: #4caf50;  
min-height: 80px;  
}  
.grid-item.nav-menu {  
background: #fff8e1;  
border-color: #ff9800;  
min-height: 80px;
```

```
}

.nav-items {
  display: flex;
  gap: 15px;
  margin-top: 10px;
}

.nav-item {
  background: #ff9800;
  color: white;
  padding: 5px 10px;
  border-radius: 4px;
  font-size: 14px;
}

.grid-item.form-field {
  background: #f8f9fa;
  border-color: #6c757d;
  min-height: 100px;
}

.demo-info {
  margin-top: 40px;
  padding: 20px;
  background: #f8f9fa;
  border: 1px solid #dee2e6;
  border-radius: 8px;
}

.demo-info h4 {
  margin: 0 0 16px 0;
  color: #333;
  font-size: 18px;
```

```
font-weight: 600;  
}  
  
.demo-info ul {  
margin: 0 0 20px 0;  
padding-left: 20px;  
color: #555;  
}  
  
.demo-info li {  
margin-bottom: 8px;  
line-height: 1.4;  
}  
  
.demo-info strong {  
color: #333;  
}  
  
,  
],  
})  
  
export class GridResponsiveComponent {  
constructor() {  
console.log("Grid Responsive Component initialized");  
}  
}
```

Flexbox Utilities

Advanced flexbox utilities for controlling layout direction, alignment, and spacing.

[Error extracting code]

Alignment

Flexbox alignment utilities for controlling how items are positioned within containers.

[Error extracting code]

Ordering

CSS order utilities for controlling the visual order of flex items.

[Error extracting code]

Features

Container System

- Fixed Container::containerwith responsive max-widths
- Fluid Container::container-fluidfor full-width layouts
- Responsive Breakpoints: Automatic width adjustments at different screen sizes
- Consistent Padding: 15px horizontal padding on all containers

Row System

- Flexbox Rows:.rowcreates flex containers with wrap enabled
- Negative Margins: Automatic gutter compensation with -15px margins
- No Gutters:.no-guttersclass removes spacing between columns
- Responsive Behavior: Consistent behavior across all breakpoints

Column System

- 12-Column Grid: Standard 12-column layout system
- Responsive Classes: Breakpoint-specific column classes (sm, md, lg, xl)
- Auto Columns:.coland.col-autofor flexible sizing
- Fixed Widths: Classes from.col-1to.col-12for precise control

Breakpoint System

- Extra Small (xs): Default, no prefix (0px and up)
- Small (sm): 576px and up
- Medium (md): 768px and up
- Large (lg): 992px and up
- Extra Large (xl): 1200px and up

Flexbox Utilities

- Direction: Row, column, and reverse variants
- Wrap: Wrap, nowrap, and wrap-reverse options

- Justify Content: Start, end, center, between, around
- Align Items: Start, end, center, baseline, stretch
- Align Content: Start, end, center, between, around, stretch
- Align Self: Auto, start, end, center, baseline, stretch

Positioning Utilities

- Offsets:.offset-*classes for column positioning
- Push/Pull:.push-*and.pull-*for reordering
- Order:.order-*classes for flex item ordering
- Responsive Variants: All utilities available at each breakpoint

CSS Classes

Container Classes

[TABLE]

Class | Description

.container | Fixed-width container with responsive max-widths

.container-fluid | Full-width container spanning the entire viewport

[/TABLE]

Row Classes

[TABLE]

Class | Description

.row | Flexbox row container with wrap enabled

.no-gutters | Removes gutters from row and its columns

[/TABLE]

Column Classes

[TABLE]

Class | Width | Description

.col | Auto | Equal-width column that grows to fill space

.col-auto | Auto | Column sized to content

.col-1 | 8.333% | 1/12 width column
.col-2 | 16.667% | 2/12 width column
.col-3 | 25% | 3/12 width column
.col-4 | 33.333% | 4/12 width column
.col-5 | 41.667% | 5/12 width column
.col-6 | 50% | 6/12 width column
.col-7 | 58.333% | 7/12 width column
.col-8 | 66.667% | 8/12 width column
.col-9 | 75% | 9/12 width column
.col-10 | 83.333% | 10/12 width column
.col-11 | 91.667% | 11/12 width column
.col-12 | 100% | Full-width column

[/TABLE]

[TABLE]

Class | Breakpoint | Description

.col-sm-* | $\geq 576px$ | Small screen columns
.col-md-* | $\geq 768px$ | Medium screen columns
.col-lg-* | $\geq 992px$ | Large screen columns
.col-xl-* | $\geq 1200px$ | Extra large screen columns

[/TABLE]

Flexbox Direction Classes

[TABLE]

Class | Description

.flex-row | Horizontal direction (default)
.flex-column | Vertical direction
.flex-row-reverse | Horizontal direction, reversed
.flex-column-reverse | Vertical direction, reversed

[/TABLE]

Flexbox Wrap Classes

[TABLE]

Class | Description

.flex-wrap | Allow items to wrap to next line

.flexnowrap | Prevent wrapping (default)

.flex-wrap-reverse | Wrap items in reverse order

[/TABLE]

Justify Content Classes

[TABLE]

Class | Description

.justify-content-start | Align items to start

.justify-content-end | Align items to end

.justify-content-center | Center items

.justify-content-between | Space items evenly

.justify-content-around | Space items with equal margins

[/TABLE]

Align Items Classes

[TABLE]

Class | Description

.align-items-start | Align items to start

.align-items-end | Align items to end

.align-items-center | Center items

.align-items-baseline | Align items to baseline

.align-items-stretch | Stretch items (default)

[/TABLE]

Align Content Classes

[TABLE]

Class | Description

.align-content-start | Align content to start
.align-content-end | Align content to end
.align-content-center | Center content
.align-content-between | Space content evenly
.align-content-around | Space content with equal margins
.align-content-stretch | Stretch content (default)

[/TABLE]

Align Self Classes

[TABLE]

Class | Description

.align-self-auto | Auto alignment (default)
.align-self-start | Align self to start
.align-self-end | Align self to end
.align-self-center | Center self
.align-self-baseline | Align self to baseline
.align-self-stretch | Stretch self

[/TABLE]

Order Classes

[TABLE]

Class | Description

.order-first | Order: -1
.order-last | Order: 13
.order-1 | Order: 1
.order-2 | Order: 2
.order-3 | Order: 3

[/TABLE]

Offset Classes

[TABLE]

Class	Margin Left	Description
.offset-1	8.333%	Offset by 1 column
.offset-2	16.667%	Offset by 2 columns
.offset-3	25%	Offset by 3 columns
.offset-4	33.333%	Offset by 4 columns
.offset-5	41.667%	Offset by 5 columns
.offset-6	50%	Offset by 6 columns
.offset-7	58.333%	Offset by 7 columns
.offset-8	66.667%	Offset by 8 columns
.offset-9	75%	Offset by 9 columns
.offset-10	83.333%	Offset by 10 columns
.offset-11	91.667%	Offset by 11 columns

Push/Pull Classes

[TABLE]

Class | Description

- .push-* | Push columns to the right
- .pull-* | Pull columns to the left

[/TABLE]

CSS Custom Properties

The Grid system uses standard CSS properties and doesn't rely on CSS custom properties. However, it's designed to work seamlessly with CSS custom properties for theming:

[TABLE]

Property | Description | Default

- grid-gutter-width | Width of grid gutters | 30px
- grid-columns | Number of grid columns | 12
- container-max-widths | Container max-widths | 540px, 720px, 960px, 1140px
- grid-breakpoints | Breakpoint values | 576px, 768px, 992px, 1200px

[/TABLE]

Best Practices

Layout Structure

- Container First: Always start with a container (.container or .container-fluid)
- Row Wrapper: Wrap columns in a .row element
- Column Classes: Use appropriate column classes for desired widths
- Responsive Design: Use breakpoint-specific classes for mobile-first design

Responsive Design

- Mobile First: Design for mobile devices first, then enhance for larger screens
- Breakpoint Strategy: Use consistent breakpoints across your application
- Progressive Enhancement: Start with basic layout, add complexity for larger screens
- Content Priority: Ensure important content is visible on all screen sizes

Performance

- CSS Efficiency: Grid classes are optimized for performance
- Minimal Nesting: Avoid deeply nested grid structures when possible
- Semantic HTML: Use semantic HTML elements with grid classes
- Browser Support: Grid system works in all modern browsers

Accessibility

- Logical Order: Ensure content flows logically regardless of visual order
- Screen Readers: Grid layout doesn't affect screen reader navigation
- Keyboard Navigation: Maintain logical tab order
- Focus Management: Ensure focus indicators are visible

Content Organization

- Consistent Spacing: Use consistent gutters and spacing
- Content Hierarchy: Use grid to establish clear visual hierarchy
- Flexible Content: Design content to work across different column widths
- Responsive Images: Ensure images scale appropriately within grid columns

Accessibility Guidelines

Semantic Structure

- HTML Semantics: Use semantic HTML elements with grid classes
- Document Outline: Ensure proper heading hierarchy within grid layouts
- Landmark Roles: Use appropriate ARIA landmarks for major layout sections
- Content Flow: Maintain logical content flow regardless of visual layout

Screen Reader Support

- Logical Order: Content should flow logically for screen readers
- Descriptive Labels: Use descriptive labels for interactive elements
- Skip Links: Provide skip links for navigation-heavy layouts
- Focus Indicators: Ensure focus indicators are visible and logical

Keyboard Navigation

- Tab Order: Maintain logical tab order through grid layouts
- Focus Management: Ensure focus moves logically through content
- Keyboard Shortcuts: Provide keyboard shortcuts for complex layouts
- Escape Mechanisms: Allow users to escape from complex layouts

Visual Design

- High Contrast: Ensure sufficient contrast ratios in grid layouts
- Color Independence: Don't rely solely on color for information
- Text Scaling: Support for text scaling and zoom
- Motion Sensitivity: Provide alternatives for users sensitive to motion

Responsive Behavior

Mobile Adaptations

The Grid system automatically adapts to mobile screens:

- Stacked Layout: Columns stack vertically on small screens
- Full Width: Single columns take full width on mobile
- Touch Optimization: Layouts are optimized for touch interaction
- Performance: Optimized performance for mobile devices

Breakpoint Behavior

- Extra Small (0px+): Single column layout, stacked content
- Small (576px+): Multi-column layouts begin to appear
- Medium (768px+): Tablet-optimized layouts
- Large (992px+): Desktop layouts with full feature set
- Extra Large (1200px+): Large desktop and wide screen layouts

Content Considerations

- Flexible Images: Images scale appropriately within grid columns
- Text Readability: Text remains readable at all column widths
- Touch Targets: Interactive elements maintain appropriate touch target sizes
- Loading Performance: Layouts load efficiently on all devices

DIVIDER

A simple and effective visual separator for organizing content sections. Features multiple styling variants, horizontal and vertical orientations, and customizable colors for clear content separation.

How to use

```
import  
{  
  AavaDividersComponent  
}  
from  
"@aava/play-core"  
;
```

Basic Usage

Simple divider implementations for content separation.

```
<aava-dividers  
variant="solid"  
orientation="horizontal"  
color="#BBBEC5"  
></aava-dividers>
```

Variants

Different divider styles for various visual contexts.

```
<aava-dividers  
variant="solid"  
orientation="horizontal"  
color="#BBBEC5"  
></aava-dividers>  
  
<aava-dividers  
variant="dashed"  
orientation="horizontal"
```

```
color="#BBBEC5"
></aava-dividers>
<aava-dividers
variant="dotted"
orientation="horizontal"
color="#BBBEC5"
></aava-dividers>
<aava-dividers variant="gradient" orientation="horizontal"></aava-dividers>
```

Variant Types

- Solid: Clean, continuous line (default)
- Dashed: Dashed line pattern
- Dotted: Dotted line pattern
- Gradient: Smooth gradient effect

Orientation

Horizontal and vertical divider orientations for different layout needs.

```
<aava-dividers
variant="solid"
orientation="vertical"
color="#dc3545"
></aava-dividers>
<aava-dividers
variant="dotted"
orientation="vertical"
color="#ffc107"
></aava-dividers>
</div>
<aava-dividers
variant="solid"
```

```
orientation="vertical"
color="#dc3545"
></aava-dividers>
<aava-dividers
variant="dotted"
orientation="vertical"
color="#ffc107"
></aava-dividers>
```

API Reference

Inputs

[TABLE]

Property | Type | Default | Description

variant | 'solid' | 'dashed' | 'dotted' | 'gradient' | 'solid' | Visual style of the divider

orientation | 'horizontal' | 'vertical' | 'horizontal' | Direction of the divider

color | string | '#000000' | Color of the divider (hex, rgb, or CSS color name)

[/TABLE]

CSS Custom Properties

[TABLE]

Property | Default | Description

--divider-color | Component color input | Dynamic color applied to borders

--divider-background-gradient | Theme-based | Gradient background for gradient variant

[/TABLE]

Best Practices

Design Guidelines

- Use dividers sparingly to avoid visual clutter
- Choose appropriate variants based on content hierarchy
- Ensure sufficient color contrast for accessibility
- Consider spacing around dividers for optimal visual separation

Layout Usage

- Use horizontal dividers between content sections
- Use vertical dividers in sidebar layouts or columns
- Apply consistent divider styles throughout the interface
- Match divider colors to your design system

Accessibility & Dev Guidelines

The Divider component is purely visual and should not convey crucial content structure on its own. However, it's accessible:

- Uses role="separator"
- Honors aria-orientation="horizontal | vertical"
- When used with text, ensure surrounding context is semantic
- Test selector: data-testid="play-divider"

Do's and Don'ts

[TABLE]

Do | Don't

Use for layout separation in dense UIs | Don't use as a replacement for headings

Style via tokens for consistent spacing | Don't hardcode padding/margins

Add labels only when useful context exists | Don't overuse labeled dividers visually

Use vertical divider inside horizontal layout | Don't nest labeled dividers unnecessarily

Use | inset | variant to separate nested blocks | Don't rely solely on divider for structure

[/TABLE]

HEADER

The Header component is a flexible, responsive layout component designed to create consistent application headers and navigation bars. It features a three-column layout system with content projection, multiple theme variants, and intelligent responsive behavior that adapts to different screen sizes.

The

```
<aava-header>
```

component provides a structured approach to building headers with left, center, and right content sections, making it ideal for logos, navigation menus, search bars, user actions, and other header elements.

How to use

```
import
{
  AavaHeaderComponent
}
from
"@aava/play-core"
;
```

Basic Usage

Simple header with logo, navigation, and user actions.

```
<aava-header containerClass="">
<div left-content class="header__left">
<div class="logo"></div>
</div>
<div center-content class="header__center tabs-header">
<aava-tabs
  [tabs]="iconsTabs"
  [activeTabId]="activeTabId"
  variant="default"
  size="sm">
```

```
[showContentPanels]="false"
(tabChange)="onTabChange($event)"
ariaLabel="Basic navigation tabs"
></aava-tabs>
</div>
<div right-content class="header__right">
<div class="search-box">
<aava-icon iconName="search" [iconSize]="16" iconColor="#000"></aava-icon>
<input
type="text"
[(ngModel)]="searchText"
(input)="onSearchChange()"
placeholder="Search"
/>
</div>
<aava-icon
class="icon"
iconName="Sun"
[iconSize]="20"
[iconColor]="getIconColor"
(click)="onGraphIconClick()"></aava-icon>
<aava-icon
class="icon"
iconName="Languages"
[iconSize]="20"
[iconColor]="getIconColor"
(click)="onTranslateIconClick()"></aava-icon>
```

```
<aava-icon
  class="icon"
  iconName="EllipsisVertical"
  [iconSize]="20"
  [iconColor]="getIconColor"
  (click)="onDayLightModelIconClick()"
></aava-icon>
<aava-avatars
  size="md"
  shape="square"
  [imageUrl]="getLoggedInUserLogo()"
  badgeState="high-priority"
  badgeSize="lg"
  [badgeCount]="9"
></aava-avatars>
</div>
</aava-header>
---
logUrl: string = "assets/asc-logo.png";
activeTabId: string = 'overview';
private baseIconsTabs: DemoTabItem[] = [
{
  id: 'tab1',
  label: 'Tab1',
  content: 'Content 1',
  iconName: 'chevron-right',
},
{
  id: 'tab2',

```

```
label: 'Tab2',
content: 'Content 2',
iconName: 'chevron-right',
},
{
id: 'tab3',
label: 'Tab3',
content: 'Content 3',
iconName: 'chevron-right',
},
{
id: 'tab4',
label: 'Tab4',
content: 'Content 4',
iconName: 'chevron-right',
},
];
accounts = [
{
companyName: 'ASCENDION',
email: 'Jhon.Stark@Ascendion.com',
companyLogo: 'assets/logos/dlogo.svg',
userLogo: 'assets/1.svg',
isLoggedIn: true,
},
{
companyName: 'Axos',
email: 'Akash.Kumar@Axos.com',
companyLogo: 'assets/logos/dlogo.svg',
```

```
userLogo: 'assets/1.svg',
isLoggedIn: false,
},
];
get iconsTabs(): TabItem[] {
return this.baselconsTabs.map((tab) => ({
...tab,
active: tab.id === this.activeTabId,
}));}
}
get getIconColor(): string {
return '#000';
}
onTabChange(event: any) {
this.activeTabId = event.id;
}
onSearchChange() {
console.log('Search text changed:', this.searchText);
}
onGraphIconClick() {
console.log('Graph Icon Clicked');
}
onTranslateIconClick() {
console.log('Translate Icon Clicked');
}
onDayLightModelIconClick() {
console.log('Day Light Mode Icon Clicked');
}
getLoggedInUserLogo(): string {
```

```
const loggedInAccount = this.accounts.find((account) => account.isLoggedIn);

return loggedInAccount ? loggedInAccount.userLogo : 'assets/1.svg';

}
```

Variations

Without Navigation

```
<aava-header containerClass="">

<div left-content class="header__left">

<div class="logo"></div>

</div>

<div right-content class="header__right">

<div class="search-box">

<aava-icon iconName="search" [iconSize]="16" iconColor="#000"></aava-icon>

<input

type="text"

[(ngModel)]="searchText"

(input)="onSearchChange()"

placeholder="Search"

/>

</div>

<aava-icon

class="icon"

iconName="Sun"

[iconSize]="20"

[iconColor]="getIconColor"

(click)="onGraphIconClick()"

></aava-icon>

<aava-icon

class="icon"
```

```
iconName="Languages"
[iconSize]="20"
[iconColor]="getIconColor"
(click)="onTranslateIconClick()"
></aava-icon>

<aava-icon
class="icon"
iconName="EllipsisVertical"
[iconSize]="20"
[iconColor]="getIconColor"
(click)="onDayLightModelIconClick()"
></aava-icon>

<aava-avatars
size="md"
shape="square"
[imageUrl]="getLoggedInUserLogo()"
badgeState="low-priority"
badgeSize="lg"
[badgeCount]="9"
></aava-avatars>
</div>
</aava-header>
---

logoUrl: string = "assets/asc-logo.png";
accounts = [
{
  companyName: 'ASCENDION',
  email: 'Jhon.Stark@Ascendion.com',
  companyLogo: 'assets/logos/dlogo.svg',
```

```
userLogo: 'assets/1.svg',
isLoggedIn: true,
},
{
companyName: 'Axos',
email: 'Akash.Kumar@Axos.com',
companyLogo: 'assets/logos/dlogo.svg',
userLogo: 'assets/1.svg',
isLoggedIn: false,
},
];
get iconsTabs(): TabItem[] {
return this.baselconsTabs.map((tab) => ({
...tab,
active: tab.id === this.activeTabId,
}));}
get getIconColor(): string {
return '#000';
}
onSearchChange() {
console.log('Search text changed:', this.searchText);
}
onGraphIconClick() {
console.log('Graph Icon Clicked');
}
onTranslateIconClick() {
console.log('Translate Icon Clicked');
}
```

```
onDayLightModelIconClick() {  
  console.log('Day Light Mode Icon Clicked');  
}  
  
getLoggedInUserLogo(): string {  
  const loggedInAccount = this.accounts.find((account) => account.isLoggedIn);  
  return loggedInAccount ? loggedInAccount.userLogo : 'assets/1.svg';  
}
```

A compact header layout that omits the navigation section—ideal for minimal pages or dashboards.

Without Search

```
<aava-header containerClass="">  
  
<div left-content class="header__left">  
  
<div class="logo"></div>  
  
</div>  
  
<div center-content class="header__center tabs-header">  
  
<aava-tabs  
  
[tabs]="iconsTabs"  
[activeTabId]="activeTabId"  
variant="default"  
size="sm"  
  
[showContentPanels]="false"  
(tabChange)="onTabChange($event)"  
ariaLabel="Basic navigation tabs"  
  
></aava-tabs>  
  
</div>  
  
<div right-content class="header__right">  
  
<aava-icon  
  
class="icon"  
iconName="Sun"  
[iconSize]="20"
```

```
[iconColor]="getIconColor"
(click)="onGraphIconClick()"
></aava-icon>
<aava-icon
class="icon"
iconName="Languages"
[iconSize]="20"
[iconColor]="getIconColor"
(click)="onTranslateIconClick()"
></aava-icon>
<aava-icon
class="icon"
iconName="EllipsisVertical"
[iconSize]="20"
[iconColor]="getIconColor"
(click)="onDayLightModelIconClick()"
></aava-icon>
<aava-avatars
size="md"
shape="square"
[imageUrl]="getLoggedInUserLogo()"
badgeState="medium-priority"
badgeSize="lg"
[badgeCount]="9"
></aava-avatars>
</div>
</aava-header>
---
logoUrl: string = "assets/asc-logo.png";
```

```
activeTabId: string = 'overview';

private baseIconsTabs: DemoTabItem[] = [
  {
    id: 'tab1',
    label: 'Tab1',
    content: 'Content 1',
    iconName: 'chevron-right',
  },
  {
    id: 'tab2',
    label: 'Tab2',
    content: 'Content 2',
    iconName: 'chevron-right',
  },
  {
    id: 'tab3',
    label: 'Tab3',
    content: 'Content 3',
    iconName: 'chevron-right',
  },
  {
    id: 'tab4',
    label: 'Tab4',
    content: 'Content 4',
    iconName: 'chevron-right',
  },
];
accounts = [
  {
```

```
    companyName: 'ASCENDION',
    email: 'Jhon.Stark@Ascendion.com',
    companyLogo: 'assets/logos/dlogo.svg',
    userLogo: 'assets/1.svg',
    isLoggedIn: true,
  },
  {
    companyName: 'Axos',
    email: 'Akash.Kumar@Axos.com',
    companyLogo: 'assets/logos/dlogo.svg',
    userLogo: 'assets/1.svg',
    isLoggedIn: false,
  },
];
get iconsTabs(): TabItem[] {
  return this.baselconsTabs.map((tab) => ({
    ...tab,
    active: tab.id === this.activeTabId,
  }));
}
get getIconColor(): string {
  return '#000';
}
onTabChange(event: any) {
  this.activeTabId = event.id;
}
onGraphIconClick() {
  console.log('Graph Icon Clicked');
}
```

```
onTranslateIconClick() {  
  console.log('Translate Icon Clicked');  
}  
  
onDayLightModelIconClick() {  
  console.log('Day Light Mode Icon Clicked');  
}  
  
getLoggedInUserLogo(): string {  
  const loggedInAccount = this.accounts.find((account) => account.isLoggedIn);  
  return loggedInAccount ? loggedInAccount.userLogo : 'assets/1.svg';  
}
```

Displays a clean header without the search bar—suitable for content-focused layouts or authenticated user dashboards.

Mobile View

```
<aava-header containerClass="">  
  
<div left-content class="header__left">  
  
<aava-icon  
  class="menu-toggle"  
  iconName="Menu"  
  [iconSize]="24"  
  iconColor="#000"  
  (click)="toggleMenuDropdown()">  
</aava-icon>  
  
<div class="menu-dropdown" [class.open]="isMenuOpen">  
  
<div class="dropdown-logo">  
    
</div>  
  
<div class="search-box dropdown-search">  
  
<aava-icon  
  iconName="search"
```

```
[iconSize]="16"
iconColor="#000"
></aava-icon>
<input
type="text"
[(ngModel)]="searchText"
(input)="onSearchChange()"
placeholder="Search"
/>
</div>
<div class="dropdown-tabs">
<aava-tabs
[tabs]="iconsTabs"
[activeTabId]="activeTabId"
variant="default"
size="sm"
[showContentPanels]="false"
orientation="vertical"
(tabChange)="onTabChange($event)"
>
</aava-tabs>
</div>
</div>
</div>
<div right-content class="header__right">
<aava-icon
class="icon"
iconName="Sun"
[iconSize]="20"
```

```
[iconColor]="getIconColor"
(click)="onGraphIconClick()"
></aava-icon>
<aava-icon
class="icon"
iconName="Languages"
[iconSize]="20"
[iconColor]="getIconColor"
(click)="onTranslateIconClick()"
></aava-icon>
<aava-avatars
size="md"
shape="square"
[imageUrl]="getLoggedInUserLogo()"
badgeState="information"
badgeSize="lg"
[badgeCount]="9"
></aava-avatars>
<aava-icon
class="icon"
iconName="EllipsisVertical"
[iconSize]="20"
[iconColor]="getIconColor"
(click)="onDayLightModelIconClick()"
></aava-icon>
</div>
</aava-header>
---
logoUrl: string = "assets/asc-logo.png";
```

```
searchText = "";

isDropdownOpen = true;

activeTabId: string = "overview";

isMenuOpen = false;

private baseIconsTabs: DemoTabItem[] = [
{
  id: 'tab1',
  label: 'Tab1',
  content: 'Content 1',
  iconName: 'chevron-right',
},
{
  id: 'tab2',
  label: 'Tab2',
  content: 'Content 2',
  iconName: 'chevron-right',
},
{
  id: 'tab3',
  label: 'Tab3',
  content: 'Content 3',
  iconName: 'chevron-right',
},
{
  id: 'tab4',
  label: 'Tab4',
  content: 'Content 4',
  iconName: 'chevron-right',
},
```

```
];
accounts = [
{
  companyName: 'ASCENDION',
  email: 'Jhon.Stark@Ascendion.com',
  companyLogo: 'assets/logos/dlogo.svg',
  userLogo: 'assets/1.svg',
  isLoggedIn: true,
},
{
  companyName: 'Axos',
  email: 'Akash.Kumar@Axos.com',
  companyLogo: 'assets/logos/dlogo.svg',
  userLogo: 'assets/1.svg',
  isLoggedIn: false,
},
];
get getIconColor(): string {
  return '#000';
}
get iconsTabs(): TabItem[] {
  return this.baselconsTabs.map((tab) => ({
    ...tab,
    active: tab.id === this.activeTabId,
  }));
}
onSearchChange() {
  console.log('Search text changed:', this.searchText);
}
```

```

toggleMenuDropdown() {
  this.isMenuOpen = !this.isMenuOpen;
}

onDayLightModelIconClick() {
  console.log('Day Light Mode Icon Clicked');
}

onTranslateIconClick() {
  console.log('Translate Icon Clicked');
}

onGraphIconClick() {
  console.log('Graph Icon Clicked');
}

onTabChange(event: any) {
  this.activeTabId = event.id;
}

getSortedAccounts() {
  return this.accounts.sort((a, b) =>
    a.isLoggedIn === b.isLoggedIn ? 0 : a.isLoggedIn ? -1 : 1
  );
}

getLoggedInUserLogo(): string {
  const loggedInAccount = this.accounts.find((account) => account.isLoggedIn);
  return loggedInAccount ? loggedInAccount.userLogo : 'assets/1.svg';
}

```

Responsive version optimized for smaller screens, with collapsed or hidden center content and accessible mobile actions.

Content Layouts

Layout Patterns

- Logo + Navigation + Actions- Traditional header with main navigation

- Logo + Search + User Menu- Application header with search functionality
- Minimal Logo + Navigation- Clean, simple header design

Responsive Behavior

Responsive Breakpoints

- Above 1440px: Full three-column layout
- Below 1440px: Center section with visual separator, right section pinned
- Below 1280px: Center section hidden, simplified mobile layout

Features

Three-Column Layout System

- Left Content: Typically for logos, brand elements, or primary navigation
- Center Content: Main navigation, search bars, or secondary content
- Right Content: User actions, notifications, or tertiary navigation

Content Projection

- Uses Angular content projection with `left-content`, `center-content`, and `right-content` selectors
- Flexible content placement without component restrictions
- Easy customization and styling

Theme System

- Light, dark, and transparent theme variants
- Consistent color schemes across themes
- Easy theme switching and customization

Responsive Design

- Automatic layout adjustments based on screen size
- Visual separators that appear/disappear appropriately
- Mobile-optimized layouts

Custom Styling

- `containerClass` input for additional CSS classes
- Flexible styling through content projection

- CSS custom properties for theme customization

API Reference

Input Properties

[TABLE]

Property | Type | Default | Description

theme | 'light' | 'dark' | 'transparent' | 'light' | Header theme variant

containerClass | string | '' | Additional CSS classes for the header container

[/TABLE]

Content Projection Selectors

[TABLE]

Selector | Description

[left-content] | Left section content (logo, brand)

[center-content] | Center section content (navigation, search)

[right-content] | Right section content (actions, user menu)

[/TABLE]

Best Practices

Content Organization

- Left Section: Place logos, brand names, or primary navigation
- Center Section: Use for main navigation menus or search functionality
- Right Section: Include user actions, notifications, or secondary navigation

Responsive Considerations

- Design for mobile-first approach
- Ensure center content gracefully hides on smaller screens
- Test layout behavior across different breakpoints

Theme Usage

- Light Theme: Standard applications, content-heavy pages
- Dark Theme: Dashboard applications, media-focused interfaces

- Transparent Theme: Hero sections, overlay headers, landing pages

Accessibility

- Ensure sufficient color contrast in all themes
- Provide clear navigation structure
- Use semantic HTML elements within content sections

Styling

The header component uses CSS Grid and Flexbox for layout management with:

- Automatic spacing and alignment
- Responsive breakpoints with CSS media queries
- Theme-based color schemes
- Visual separators for center content
- Smooth transitions and animations

PROGRESS-BAR

The

```
<aava-progressbar>
```

provides a comprehensive progress indicator solution with both circular and linear variants. It supports multiple modes including determinate, indeterminate, buffer, and query states, with smooth animations, customizable colors, and full accessibility support.

How to use

Import the component and configure it with your desired properties.

```
import  
{  
    AavaProgressComponent  
}  
from  
"@aava/play-core"  
;
```

Circular Progress

Circular progress indicator with customizable size, color, and animation.

```
<aava-progressbar  
[percentage] = "25"  
label = "25% Complete"  
type = "circular"  
[svgSize] = "100"  
>  
</aava-progressbar>  
<aava-progressbar  
[percentage] = "50"  
label = "50% Complete"  
type = "circular"  
[svgSize] = "100"
```

```
>  
</aava-progressbar>  
  
<aava-progressbar  
[percentage]:"75"  
label="75% Complete"  
type="circular"  
[svgSize]="100"  
  
>  
</aava-progressbar>  
  
<aava-progressbar  
[percentage]:"100"  
label="100% Complete"  
type="circular"  
[svgSize]="100"  
  
>  
</aava-progressbar>
```

Linear Progress

Linear progress bar with support for determinate, indeterminate, and buffer modes.

```
<aava-progressbar [percentage]:"25" type="linear" mode="determinate">  
</aava-progressbar>  
  
<aava-progressbar [percentage]:"50" type="linear" mode="determinate">  
</aava-progressbar>  
  
<aava-progressbar [percentage]:"75" type="linear" mode="determinate">  
</aava-progressbar>  
  
<aava-progressbar [percentage]:"100" type="linear" mode="determinate">  
</aava-progressbar>
```

Features

Multiple Progress Types

- Circular Progress: SVG-based circular indicator with smooth animations
- Linear Progress: Horizontal progress bar with customizable height and styling
- Responsive Design: Automatically adjusts size based on screen dimensions

Progress Modes

- Determinate: Shows exact progress percentage with smooth animations
- Indeterminate: Animated loading indicator for unknown progress
- Buffer: Shows both progress and buffer values (linear only)
- Query: Animated indicator for querying operations

Customization Options

- Custom colors and themes
- Configurable sizes and positions
- Smooth animations with easing
- Accessibility features with ARIA support

Performance Optimized

- OnPush change detection strategy
- Efficient SVG animations
- Optimized rendering and memory management
- Responsive resize handling

API Reference

Inputs

[TABLE]

Property | Type | Default | Description

percentage | number | 0 | Progress percentage (0-100)

bufferValue | number | 0 | Buffer value for buffer mode (0-100)

label | string | "" | Label text displayed with the progress bar

type | 'circular' | 'linear' | 'circular' | Type of progress indicator

color | string | '#2E308E' | Color of the progress indicator

mode | 'determinate' | 'indeterminate' | 'buffer' | 'query' | 'determinate' | Progress mode

svgSize | number | '' | Custom SVG size (overrides responsive sizing)
position | '12' | '3' | '6' | '9' | number | '12' | Starting position for circular progress (clock positions)
[/TABLE]

Outputs

[TABLE]

Property	Type	Description
None	-	This component doesn't emit events

[/TABLE]

Properties

[TABLE]

Property	Type	Description
progressId	string	Unique identifier for the progress element
circumference	number	Circumference of the circular progress (readonly)
dashOffset	number	Current dash offset for circular progress
errorMessage	string	Error message for invalid inputs
displayPercentage	number	Animated display percentage for linear progress
rotationAngle	number	Rotation angle for circular progress position

[/TABLE]

Methods

[TABLE]

Method	Parameters	Description
updateProgress()	None	Updates the progress display and animations
writeValue()	value: number	Sets the progress value (ControlValueAccessor)
registerOnChange()	fn: (value: number) => void	Registers change callback (ControlValueAccessor)
registerOnTouched()	fn: () => void	Registers touched callback (ControlValueAccessor)

[/TABLE]

CSS Custom Properties

The component uses CSS custom properties for dynamic styling:

Circular Progress Properties

[TABLE]

Property | Description

--progress-text-weight Font weight for progress text
--progress-text-color Text color for progress display
--progress-text-font Font size for progress text
--progress-label-font Font size for progress label
--progress-label-color Color for progress label
--progress-label-line-height Line height for progress label
--progress-label-weight Font weight for progress label
--progress-transition Transition timing for progress animations
--progress-indeterminate-animation Animation for indeterminate mode

[/TABLE]

Linear Progress Properties

[TABLE]

Property | Description

--progress-linear-height Height of the linear progress bar
--progress-linear-border-radius Border radius for linear progress

[/TABLE]

CSS Classes

The component uses CSS classes for styling and state management:

Container Classes

[TABLE]

Class | Description

.progress-container Main container for circular progress
.linear-progress-container Main container for linear progress

[/TABLE]

Progress Classes

[TABLE]

Class	Description
.progress-background	Background circle for circular progress
.progress-bar	Main progress circle/bar
.progress-text	Text display inside circular progress
.progress-label	Label text for progress
.linear-bar	Container for linear progress bar
.linear-progress	Main linear progress element
.buffer-bar	Buffer progress element
.indeterminate-bar	Indeterminate progress element
.progress-percentage	Percentage display for linear progress

[/TABLE]

State Classes

[TABLE]

Class	Description
.indeterminate	Indeterminate animation state
.progress-error	Error message styling

[/TABLE]

Best Practices

Progress Mode Selection

- Use `determinate` mode when you know the exact progress
- Use `indeterminate` mode for unknown loading times
- Use `buffer` mode for operations with both progress and buffer (e.g., video loading)
- Use `query` mode for search or query operations

Color and Styling

- Choose colors that provide good contrast with the background
- Use consistent colors across your application
- Consider using semantic colors (success, warning, error) for different states

- Ensure accessibility with proper color contrast ratios

Performance

- Avoid frequent percentage updates for smooth animations
- Use appropriate animation durations based on the operation
- Consider using indeterminate mode for very short operations

Accessibility

- Always provide meaningful labels
- The component includes proper ARIA attributes
- Ensure sufficient color contrast
- Test with screen readers

Responsive Design

- The component automatically adjusts size on different screens
- Test on various device sizes
- Consider custom sizing for specific use cases

Accessibility

ARIA Support

- Proper role="progressbar" attribute
- aria-valuenow for current progress value
- aria-valuemin="0" and aria-valuemax="100" for value range
- Screen reader announcements for progress updates

Keyboard Navigation

- Focus indicators for interactive elements
- Proper tab order
- Keyboard-accessible progress controls

Visual Accessibility

- High contrast color options
- Clear visual indicators

- Scalable text and graphics
- Support for reduced motion preferences

Browser Support

- Modern Browsers: Full support for all features
- SVG Support: Required for circular progress
- CSS Animations: Required for smooth transitions
- ES6+ Features: Required for component functionality
- Change Detection: OnPush strategy support

SPINNER

A

Spinner

is a loading indicator that provides visual feedback to users during asynchronous operations. It helps maintain user engagement by clearly communicating that a process is in progress, reducing perceived wait times and improving user experience.

Import

```
import
{
  AavaSpinnerComponent
}
from
"@aava/play-core"
;
```

Basic Usage

```
<aava-spinner
  type="circular"
  color="primary"
  size="lg"
  [animation]="true"
></aava-spinner>
```

The spinner component supports multiple visual styles, sizes, and colors to match your application's design system.

Sizes

```
<aava-spinner
  type="circular"
  color="primary"
  size="xs"
  [animation]="true"
```

```
></aava-spinner>

<aava-spinner
  type="circular"
  color="primary"
  size="sm"
  [animation]="true"
></aava-spinner>

<aava-spinner
  type="circular"
  color="primary"
  size="md"
  [animation]="true"
></aava-spinner>

<aava-spinner
  type="circular"
  color="primary"
  size="lg"
  [animation]="true"
></aava-spinner>

<aava-spinner
  type="circular"
  color="primary"
  size="xl"
  [animation]="true"
></aava-spinner>
```

Available spinner sizes:

- xs (Extra Small): Smallest spinner for inline use
- sm (Small): Compact spinner for inline use
- md (Medium): Default size for general use

- lg (Large): Prominent spinner for important operations
- xl (Extra Large): Maximum size for high-impact loading states

Colors

```
<aava-spinner  
type="circular"  
color="primary"  
size="lg"  
[animation]="true"  
></aava-spinner>  
  
<aava-spinner  
type="circular"  
color="secondary"  
size="lg"  
[animation]="true"  
></aava-spinner>  
  
<aava-spinner  
type="circular"  
color="success"  
size="lg"  
[animation]="true"  
></aava-spinner>  
  
<aava-spinner  
type="circular"  
color="warning"  
size="lg"  
[animation]="true"  
></aava-spinner>  
  
<aava-spinner  
type="circular"
```

```
color="danger"  
size="lg"  
[animation]="true"  
></aava-spinner>
```

Semantic color variants:

- primary: Default brand color
- secondary: Secondary brand color
- success: Success state indication
- warning: Warning state indication
- danger: Error or critical state indication

Accessibility

Built-in accessibility features ensuring inclusive user experience for loading states.

Accessibility Features

- ARIA Labels: Use aria-label or aria-labelledby to describe the loading state
- Live Regions: Announce loading state changes to screen readers using aria-live
- Focus Management: Ensure proper focus handling during loading states
- Reduced Motion: Respect user preferences for reduced motion with prefers-reduced-motion
- Timeout Handling: Provide fallback mechanisms for extended loading times
- Screen Reader Support: Semantic HTML structure for assistive technologies
- Keyboard Navigation: Maintain keyboard accessibility during loading states
- Color Contrast: Ensure sufficient contrast for all spinner variants
- Status Communication: Clear communication of loading progress and completion

API Reference

Inputs

[TABLE]

Property | Type | Default | Description

type | SpinnerType | 'circular' | Visual style of the spinner

size SpinnerSize 'md'	Size of the spinner
color SpinnerColor 'primary'	Color variant of the spinner
animation boolean true	Whether to animate the spinner
progressIndex number undefined	Progress value for determinate loading (0-100)

[/TABLE]

CSS Custom Properties

[TABLE]

Property Description Default
--spinner-size-xs Extra small spinner dimensions 16px
--spinner-size-sm Small spinner dimensions 20px
--spinner-size-md Medium spinner dimensions 24px
--spinner-size-lg Large spinner dimensions 48px
--spinner-size-xl Extra large spinner dimensions 64px

[/TABLE]

[TABLE]

Property Description Default
--spinner-primary-track Primary spinner track color rgba(59, 130, 246, 0.2)
--spinner-primary-fill Primary spinner fill color rgb(59, 130, 246)
--spinner-secondary-track Secondary spinner track color rgba(107, 114, 128, 0.2)
--spinner-secondary-fill Secondary spinner fill color rgb(107, 114, 128)
--spinner-success-track Success spinner track color rgba(34, 197, 94, 0.2)
--spinner-success-fill Success spinner fill color rgb(34, 197, 94)
--spinner-warning-track Warning spinner track color rgba(245, 158, 11, 0.2)
--spinner-warning-fill Warning spinner fill color rgb(245, 158, 11)
--spinner-error-track Error spinner track color rgba(239, 68, 68, 0.2)
--spinner-error-fill Error spinner fill color rgb(239, 68, 68)

[/TABLE]

[TABLE]

Property Description Default

```
--spinner-animation-duration | Spinner rotation duration | 3s  
--spinner-animation-timing | Spinner animation timing function | linear  
[/TABLE]
```

Best Practices

Design Guidelines

- Context Appropriate: Use appropriate sizes for the context and available space
- Semantic Colors: Choose colors that align with your design system and semantic meaning
- Prominent Placement: Position spinners prominently for critical operations
- Loading States: Consider using skeleton screens for complex loading states
- Clear Context: Provide clear context about what is loading
- Consistent Timing: Use consistent animation timing across your application
- Visual Hierarchy: Size spinners according to the importance of the loading operation

Performance

- Timing Thresholds: Only show spinners for operations that take more than 200ms
- Animation Optimization: Use CSS animations instead of JavaScript for better performance
- Progress Indicators: Use progress mode for operations with known duration
- State Management: Implement proper loading state management to prevent flickering
- Resource Efficiency: Avoid unnecessary re-renders during loading states
- Bundle Size: Consider lazy loading spinner variants not immediately needed

User Experience

- Clear Messaging: Provide clear messaging about what is loading
- Consistent Patterns: Use consistent spinner styles throughout your application
- Cancel Options: Consider providing cancel options for long-running operations
- Error Handling: Implement proper error handling for failed operations
- Progress Feedback: For long operations, show progress or estimated time
- Completion States: Provide clear indication when loading is complete

SKELETON-LOADER

The

```
<aava-skeleton>
```

component is a versatile loading placeholder that provides smooth animations and multiple shape options to create engaging loading states. It helps improve perceived performance by showing users that content is loading, reducing perceived wait times and providing visual feedback during data fetching operations.

How to use

```
import
{
  AavaSkeletonComponent
}
from
"@aava/play-core"
;
```

Basic Usage

A simple skeleton with default rectangle shape and wave animation.

Shapes

The skeleton component supports multiple shapes for different content types.

Custom Styling

Customize the skeleton with different colors, sizes, and background colors.

Card Skeleton

Create a complete card skeleton with multiple elements.

List Skeleton

Build list skeletons for data tables and content lists.

Features

Multiple Shapes

- Rectangle: Default shape for text lines and content blocks
- Circle: Perfect for avatars and profile pictures
- Rounded: Soft corners for modern UI elements
- Square: Sharp corners for structured content

Animation Types

- Wave: Smooth shimmer effect that moves across the skeleton
- Pulse: Gentle fade in/out effect for subtle loading states

Customization

- Flexible Sizing: Custom width and height for any content type
- Color Control: Customizable background and animation colors
- Responsive Design: Adapts to different screen sizes
- Performance Optimized: Lightweight animations with minimal impact

Accessibility

- Screen Reader Support: Proper ARIA attributes for loading states
- Reduced Motion: Respects user's motion preferences
- High Contrast: Works with high contrast mode settings
- Focus Management: Proper focus handling during loading

API Reference

Inputs

[TABLE]

Property | Type | Default | Description

width | string | '100%' | Width of the skeleton element

height | string | '20px' | Height of the skeleton element

shape | ShimmerShape | 'rectangle' | Shape of the skeleton element

animation | ShimmerAnimation | 'wave' | Animation type for the skeleton

backgroundColor | string | '#e0e0e0' | Background color of the skeleton

skeletonType | 'tableList' | 'table' | 'tableList' | Type of skeleton layout

rows | number | 5 | Number of rows in the skeleton layout

columns | number | 5 | Number of columns in the skeleton layout

isLoading | boolean | true | Whether to show the skeleton

[/TABLE]

Types

type

ShimmerShape

=

"rectangle"

|

"circle"

|

"rounded"

|

"square"

;

type

ShimmerAnimation

=

"wave"

|

"pulse"

;

CSS Classes

The component provides several CSS classes for styling:

[TABLE]

Class Name | Description

.shimmer-container | Main skeleton container

.shimmer-item | Base skeleton element

.shimmer-rectangle | Rectangle shape styling

.shimmer-circle | Circle shape styling
.shimmer-rounded | Rounded shape styling
.shimmer-square | Square shape styling
.shimmer-animation-wave | Wave animation styling
.shimmer-animation-pulse | Pulse animation styling

[/TABLE]

CSS Custom Properties

The component uses CSS custom properties for theming:

[TABLE]

Property | Description

--skeleton-border-radius | Border radius for rectangle shape
--skeleton-rounded-radius | Border radius for rounded shape
--skeleton-wave-duration | Duration of wave animation
--skeleton-pulse-duration | Duration of pulse animation
--skeleton-wave-opacity | Opacity for wave animation
--skeleton-pulse-opacity | Opacity for pulse animation
--skeleton-background-color | Background color of skeleton
--skeleton-gradient-color-start | Start color for gradient animation
--skeleton-gradient-color-end | End color for gradient animation
--skeleton-animation-timing | Timing function for skeleton animations
--skeleton-animation-easing | Easing function for skeleton transitions

[/TABLE]

Best Practices

Content Matching

- Match Content Size: Make skeleton dimensions match the actual content
- Use Appropriate Shapes: Choose shapes that represent the actual content
- Maintain Layout: Keep skeleton layout consistent with loaded content
- Consider Spacing: Include proper spacing between skeleton elements

Performance

- Limit Skeleton Count: Don't show too many skeletons at once
- Use Appropriate Duration: Keep animations smooth but not distracting
- Consider Motion Preferences: Respect user's motion preferences
- Optimize for Mobile: Ensure good performance on mobile devices

User Experience

- Show Loading State: Always indicate when content is loading
- Provide Context: Use skeletons that give users an idea of what's coming
- Smooth Transitions: Ensure smooth transition from skeleton to content
- Consistent Timing: Keep skeleton duration consistent across the app

Accessibility

- Screen Reader Support: Provide proper loading announcements
- Reduced Motion: Support users who prefer reduced motion
- High Contrast: Ensure visibility in high contrast mode
- Focus Management: Handle focus properly during loading states

Accessibility Guidelines

Screen Reader Support

- Loading Announcements: Provide clear loading state announcements
- Content Description: Describe what content is loading
- Progress Indication: Indicate loading progress when possible
- State Changes: Announce when content finishes loading

Motion and Animation

- Reduced Motion: Respect prefers-reduced-motion media query
- Animation Duration: Keep animations smooth but not distracting
- Motion Alternatives: Provide alternatives for users who can't see animations
- Performance: Ensure animations don't cause performance issues

Visual Design

- High Contrast: Ensure visibility in high contrast mode

- Color Independence: Don't rely solely on color for information
- Focus Indicators: Provide clear focus indicators
- Consistent Styling: Maintain consistent skeleton styling

Keyboard Navigation

- Focus Management: Handle focus properly during loading
- Tab Order: Maintain logical tab order
- Skip Links: Provide skip links for keyboard users
- Loading States: Indicate loading state to keyboard users

Responsive Behavior

Mobile Adaptations

The skeleton component automatically adapts to mobile screens:

- Touch Optimization: Optimized for touch interactions
- Viewport Adaptation: Adapts to different mobile viewport sizes
- Performance: Optimized performance for mobile devices
- Battery Consideration: Efficient animations for battery life

Breakpoint Behavior

- Desktop (>768px): Full skeleton with all features
- Mobile ($\leq 768\text{px}$): Optimized skeleton for mobile screens
- Content Scaling: Skeleton scales appropriately with content
- Animation Performance: Optimized animations for different devices

Content Considerations

- Flexible Sizing: Skeleton adapts to different content sizes
- Layout Preservation: Maintains layout consistency across devices
- Loading States: Consistent loading experience across platforms
- Performance: Efficient rendering on all device types

SNACKBAR

A flexible, accessible Snackbar component for transient notifications, actions, and feedback. Supports multiple positions, variants, icons, actions, and theming. Use snackbars to provide brief messages about app processes, confirmations, or errors, with optional actions.

How to use

```
import  
{  
  SnackbarService  
}  
from  
"@aava/play-core"  
;
```

Basic Usage

Show a simple Snackbar with a message.

```
import { Component, inject } from "@angular/core";  
  
import { CommonModule } from "@angular/common";  
  
import { SnackbarService, AavaButtonComponent } from "@aava/play-core";  
  
@Component({  
  standalone: true,  
  imports: [CommonModule, AavaButtonComponent],  
  template: `  
    <div>  
      <aava-button  
        label="Show Snackbar"  
        variant="primary"  
        size="md"  
        state="default"  
        (click)="showSnackbar()">  
    </div>  
  `  
})
```

```

</aava-button>
</div>
` ;
})
export class SnackbarBasicUsageDemoComponent {
private snackbar = inject(SnackbarService);
showSnackbar() {
this.snackbar.show("This is a snackbar message", "top-center");
}
}

```

Positions

Display snackbars in different screen positions.

```

import { Component, inject } from "@angular/core";
import { CommonModule } from "@angular/common";
import { SnackbarService, SnackbarPosition } from "@aava/play-core";
import { AavaButtonComponent } from "@aava/play-core";
@Component({
standalone: true,
imports: [CommonModule, AavaButtonComponent],
template: `
<div>
<aava-button
*ngFor="let pos of positions"
[label]="pos"
variant="primary"
size="md"
state="default"
(click)="showSnackbar(pos)"
>

```

```
</aava-button>

</div>
`,
})

export class SnackbarPositionsDemoComponent {
  private snackbar = inject(SnackbarService);
  positions: SnackbarPosition[] = [
    "top-left",
    "top-center",
    "top-right",
    "bottom-left",
    "bottom-center",
    "bottom-right",
    "center",
  ];
  showSnackbar(position: SnackbarPosition) {
    this.snackbar.show(`Snackbar at ${position}`, position);
  }
}
```

There are 7 common positions used to describe snackbars placement.

- top-left: Positioned at the upper-left corner.
- top-center: Positioned at the top edge, horizontally centered.
- top-right: Positioned at the upper-right corner.
- bottom-left: Positioned at the lower-left corner.
- bottom-center: Positioned at the bottom edge, horizontally centered.
- bottom-right: Positioned at the lower-right corner.
- center: Positioned exactly in the middle, both vertically and horizontally centered.

Actions & Icons

Add action buttons, icons, make snackbars dismissible or persistent.

```
import { Component, inject } from "@angular/core";
import { CommonModule } from "@angular/common";
import { SnackbarService, AavaButtonComponent } from "@aava/play-core";
@Component({
  standalone: true,
  imports: [CommonModule, AavaButtonComponent],
  template: `
    <div>
      <aava-button
        label="Snackbar with Action"
        variant="primary"
        size="md"
        state="default"
        (click)="showActionSnackbar()"
      >
        </aava-button>
    </div>
  `,
})
export class SnackbarActionsIconsDemoComponent {
  private snackbar = inject(SnackbarService);
  showActionSnackbar() {
    this.snackbar.show("File uploaded!", "bottom-center", 4000, "", "", {
      action: {
        text: "Undo",
        color: "primary",
        callback: () => alert("Undo clicked!"),
      },
    });
  }
}
```

```
});  
}  
}
```

API Reference

Inputs

[TABLE]

Property | Type | Default | Description

message | string | "" | Message text to display in the snackbar

position | SnackbarPosition | 'bottom-center' | Position of the snackbar on screen

duration | number | 4000 | Duration in milliseconds before auto-dismiss

color | string | "" | Custom text color for the snackbar

backgroundColor | string | "" | Custom background color for the snackbar

action | SnackbarAction | "" | Action button configuration

icon | SnackbarIcon | "" | Icon configuration for the snackbar

dismissible | boolean | false | Whether the snackbar can be dismissed

persistent | boolean | false | Whether the snackbar persists until dismissed

variant | 'surface-bold' | string | 'surface-bold' | Visual variant of the snackbar

type | 'medium' | 'strong' | 'max' | string | 'medium' | Size/emphasis type of the snackbar

[/TABLE]

Outputs

[TABLE]

Event | Type | Description

dismissed | EventEmitter<void> | Emitted when snackbar is dismissed

actioned | EventEmitter<string> | Emitted when action button is clicked

[/TABLE]

Methods

[TABLE]

Method | Parameters | Return Type | Description

show() | message: string, options?: SnackbarOptions | void | Display a snackbar with the given message

dismiss() | void | void | Dismiss the currently displayed snackbar

clear() | void | void | Clear all displayed snackbars

[/TABLE]

CSS Custom Properties

[TABLE]

Property | Description

--snackbar-font-family | Font family for snackbar text

--snackbar-size-sm-font | Font size for small snackbar

--snackbar-size-md-font | Font size for medium snackbar

--snackbar-size-lg-font | Font size for large snackbar

--snackbar-success-color | Color for success snackbar

--snackbar-warning-color | Color for warning snackbar

--snackbar-error-color | Color for error snackbar

--snackbar-info-color | Color for info snackbar

--snackbar-font-weight-regular | Regular font weight

--snackbar-font-weight-semibold | Semibold font weight

--snackbar-subtitle-color | Color for subtitle text

--snackbar-title-color | Color for title text

--snackbar-radius-sm | Border radius for snackbar

--snackbar-shadowbox | Box shadow for snackbar

--snackbar-padding-small | Padding for small snackbar

--snackbar-padding-medium | Padding for medium snackbar

--snackbar-padding-large | Padding for large snackbar

--snackbar-padding-default | Default padding for snackbar

--snackbar-btn-font-sm | Font size for small button

--snackbar-padding-small-btn | Padding for small button

--snackbar-btn-font-md | Font size for medium button

```
--snackbar-padding-md-btn | Padding for medium button  
--snackbar-btn-font-large | Font size for large button  
--snackbar-font-small | Small font size  
--snackbar-font-medium | Medium font size  
--snackbar-font-large | Large font size  
--snackbar-font-large-title | Font size for large title  
--snackbar-font-title | Font size for title  
--snackbar-font-subtitle | Font size for subtitle  
--snackbar-padding-wrapper | Horizontal padding for snackbar container
```

[/TABLE]

Accessibility

- Keyboard accessible (focus, dismiss)
- Proper ARIA attributes for screen readers
- Focus management and visible indicators
- High contrast and reduced motion support

Theming & Design Tokens

All colors, spacing, and effects are controlled via semantic design tokens and CSS custom properties. Override these in your theme or component styles for custom branding.

Best Practices

- Use snackbars for brief, non-blocking feedback
- Avoid placing critical information solely in snackbars
- Provide clear, actionable messages
- Use actions for undo or retry scenarios
- Test snackbar placement in responsive layouts
- Avoid excessive custom styling; use built-in variants and types