# A Comprehensive Technical Reference for the Zalo Mini App UI (ZAUI) Library

## Part I: Foundational Concepts and Environment Setup

This section establishes the foundational knowledge required to develop applications within the Zalo Mini App ecosystem. It begins with a high-level overview of the ZAUI library and its governing design principles, then transitions to the practical steps of configuring a development environment and understanding the canonical project structure.

### Chapter 1: Introduction to the Zalo Mini App Ecosystem

#### 1.1. Defining ZAUI and its Role

The Zalo Mini App UI (ZAUI) is a comprehensive library of User Interface (UI) components specifically engineered for the Zalo Mini App platform.1 Its primary objective is to provide developers with a standardized, mobile-first toolkit that accelerates the application development lifecycle.2 The components are designed and optimized to ensure a consistent and native-like user experience within the Zalo environment.

ZAUI functions as the presentation layer within a broader, more extensive Zalo Mini App ecosystem. This ecosystem provides a complete suite of tools and services for building, deploying, and managing Mini Apps. Beyond the UI components, the platform includes:

* **Platform API (ZMP-SDK):** A set of Application Programming Interfaces that allow a Mini App to interact with the Zalo platform and native device functionalities, such as accessing user information, device hardware (camera, location), and Zalo-specific features (sharing, following Official Accounts).3
* **Payment SDK:** Specialized tools for integrating various payment gateways, facilitating in-app transactions.4
* **Developer Tools (DevTools):** A suite of tools, including a command-line interface (CLI), a dedicated IDE (Zalo Mini App Studio), and extensions for popular code editors like Visual Studio Code, designed to streamline the development and debugging process.4
* **Open APIs:** A collection of APIs that enable the automation of management and development tasks for large-scale or multiple Mini App deployments.4

The relationship between ZAUI and the rest of the ecosystem is symbiotic. ZAUI provides the visual elements (the "what"), while the Platform API provides the logic and functionality (the "how"). For instance, a Button component from ZAUI is the visual element a user interacts with, and its onClick event handler would typically invoke a function from the Platform API, such as getPhoneNumber() or scanQRCode().3

#### 1.2. Core Design Principles

The architecture and aesthetic of all ZAUI components are guided by a set of core design principles that ensure applications are intuitive, efficient, and consistent with the Zalo user experience. Developers building with ZAUI are expected to adhere to these foundational guidelines.7

1. **Be Friendly and Fast:** The primary directive is to maintain simplicity in all user interactions. The user flow should be straightforward, avoiding unnecessary complexity or interruptions. This principle encourages clear, concise interfaces and performant components that respond quickly to user input.
2. **Highlight the Focal Point:** Each screen or view within a Mini App should have a single, clear point of focus. The most critical information or primary action must be visually prominent, allowing users to grasp the purpose of the screen and take action efficiently. Ancillary information and secondary actions should be presented in a more subdued, organized manner to avoid distracting from the main objective. This principle directly manifests in the component architecture itself. For example, the Button component is offered in distinct Primary, Secondary, and Tertiary visual levels.6 This is not an arbitrary styling choice; it is a deliberate implementation of this guideline. A  
   Primary button is designed with high visual weight to serve as the main focal point for action on a screen, and design rules for Button Groups stipulate that only one primary button should be present in a single group to maintain a clear action hierarchy.6
3. **Optimize the User Experience Flow:** The application's structure should be optimized to guide the user to their goal as quickly as possible. This involves eliminating extraneous steps, removing irrelevant components from critical workflows, and ensuring that the path to task completion is logical and frictionless.

Adherence to these principles is crucial for passing the platform's review process (Chính sách kiểm duyệt) and for creating successful, user-centric Mini Apps.4

### Chapter 2: Development Environment and Project Initialization

#### 2.1. Development Workflow Options

The Zalo Mini App platform offers developers flexibility in their choice of tools for project initialization and development. There are three primary, officially supported workflows, each catering to different developer preferences and project requirements.5

* **Using Zalo Mini App Extension for Visual Studio Code:** This is a powerful option for developers who prefer the feature-rich environment of VS Code. The extension integrates Zalo Mini App project management directly into the editor. The workflow involves installing the extension, using the command palette to create a new project, and selecting a ZAUI template (e.g., ZaUI Coffee). Once the project is generated, developers can configure the App ID, install dependencies, and use the integrated run panel to start the development server.
* **Using Zalo Mini App Studio:** This is a standalone Integrated Development Environment (IDE) tailored specifically for Zalo Mini App development. It provides a more guided, all-in-one experience. The process involves installing the Studio, creating a new project, entering the Mini App ID, and choosing a ZAUI template. The Studio handles the project scaffolding and provides simple "Start" button functionality to run the application.
* **Using Zalo Mini App Command-Line Interface (CLI):** This method is ideal for developers who prefer a terminal-based workflow, as well as for integrating with automated build systems and Continuous Integration/Continuous Deployment (CI/CD) pipelines. The setup requires Node.js and the installation of the zmp-cli package. A project can be started by cloning a template repository, installing dependencies with npm install, and running the development server with zmp start.

#### 2.2. Library Installation and Usage

The ZAUI component library is distributed as a standard Node.js package named zmp-ui.8 To use the components in a project, this package must be added as a dependency.

**Installation:**

Bash

npm install zmp-ui

or

Bash

yarn add zmp-ui

Component Usage:

Once installed, individual components can be imported into React component files using ES6 module syntax. For example, to use the Button component:

JavaScript

import React from "react";  
import { Button } from "zmp-ui";  
  
const MyComponent = () => {  
 return <Button size="large">Click Me</Button>;  
};  
  
export default MyComponent;

9

Stylesheet Integration:

For the components to render with the correct Zalo Mini App styling, their corresponding CSS must be included in the application's entry point (e.g., App.js). Developers have two options:

1. **Import the Global Stylesheet:** This is the simplest method and includes styles for all components in the library.  
   JavaScript  
   import 'zmp-ui/zaui.css';
2. **Import Component-Specific Stylesheets:** For projects where bundle size is a critical concern, it is possible to import styles only for the components being used. This can lead to a smaller final asset size.  
   JavaScript  
   // Example: Import styles only for the Button component  
   import 'zmp-ui/button/styles/button.css';  
     
   10

#### 2.3. Project Structure Analysis (Based on zaui-coffee Template)

The official zaui-coffee template provides a prescriptive and well-architected blueprint for structuring a Zalo Mini App project.5 Analyzing this structure reveals the platform's recommended best practices for code organization, state management, and configuration.

The canonical folder structure is as follows:

* app-config.json: The global configuration file for the Mini App. This file controls properties such as the app's title and status bar color.12
* mock/: This directory contains static JSON files used for mock data during development, allowing the UI to be built and tested independently of a live backend API.5
* src/: The main source code directory containing all application logic.
  + app.ts: The primary entry point of the Mini App.
  + components/: Contains reusable, stateless, or stateful React components that are used across multiple pages (e.g., a custom card, a product item).
  + css/: Holds stylesheets, with support for pre-processors like SCSS.
  + global.d.ts: A TypeScript declaration file for defining types for third-party modules or global objects.
  + pages/: Contains components that represent entire screens or views. Each page is typically registered as a route in the application's router.
  + state.ts: This file is dedicated to global state management. The zaui-coffee template's inclusion of this file and its specified use for "containing Recoil's atoms and selectors" is a strong architectural recommendation.5 It indicates that for non-trivial applications, a structured global state management library like Recoil is the preferred solution over prop-drilling or component-level state alone. This is a significant architectural pattern that developers should adopt for robust state handling.
  + statics/: For static assets like images and SVGs that should be bundled with the application.
  + types/: Contains TypeScript type and interface definitions for the application's data models (e.g., Product, User).
  + utils/: A directory for pure, reusable utility functions that encapsulate business logic, data transformations, or common calculations (e.g., date formatting, distance calculation).5

This structure promotes a clean separation of concerns, making the application easier to maintain, scale, and test.

## Part II: The ZAUI Design System

The ZAUI Design System is the set of fundamental rules, constraints, and primitives that define the visual language of the Zalo Mini App platform. It governs everything from typography and color to spacing and elevation, ensuring a cohesive and predictable user experience across all applications.

### Chapter 3: Typography System

#### 3.1. Font Families

ZAUI intentionally forgoes a custom font in favor of leveraging the native system fonts of the user's device. This decision ensures that text within a Mini App renders in a familiar, legible, and platform-consistent manner, enhancing the feeling of a native application.13

The specific font families used are:

* **iOS:** SF Pro Text, SF Pro Display, SF UI Text, SF UI Display
* **Android:** Roboto

#### 3.2. Text Styles and Specifications

The typography system is built on a structured scale divided into two main categories: Header Text and Body Text. Each style within these categories has precisely defined properties for font size, line height, and font weight, with the system using a weight of 400 for regular text and 500 for medium (semi-bold) text.13

**Header Text** styles are intended for prominent text elements such as page titles, dialog headers, and banners. **Body Text** styles are used for all other content, from paragraphs to labels and captions.

The complete typography scale is detailed in the table below.

| Style Name | Category | Font Size (px) | Line Height (px) | Font Weight |
| --- | --- | --- | --- | --- |
| xLarge | Header Text | 22 | 26 | 500 |
| Normal | Header Text | 18 | 24 | 500 |
| Small | Header Text | 15 | 20 | 500 |
| xLarge | Body Text | 18 | 24 | 400 |
| xLarge M | Body Text | 18 | 24 | 500 |
| Large | Body Text | 16 | 22 | 400 |
| Large M | Body Text | 16 | 22 | 500 |
| Normal | Body Text | 15 | 20 | 400 |
| Normal M | Body Text | 15 | 20 | 500 |
| Small | Body Text | 14 | 18 | 400 |
| Small M | Body Text | 14 | 18 | 500 |
| xSmall | Body Text | 13 | 18 | 400 |
| xSmall M | Body Text | 13 | 18 | 500 |
| xxSmall | Body Text | 12 | 16 | 400 |
| xxSmall M | Body Text | 12 | 16 | 500 |
| xxxSmall | Body Text | 11 | 16 | 400 |
| xxxSmall M | Body Text | 11 | 16 | 500 |
| xxxxSmall | Body Text | 10 | 14 | 400 |
| xxxxSmall M | Body Text | 10 | 14 | 500 |

### Chapter 4: Color Architecture

#### 4.1. Color Palettes

The ZAUI color system is comprehensive, providing a wide range of primary, secondary, and neutral colors to create visually rich and accessible interfaces. The system is based on a set of 10 foundational color palettes, a full neutral (gray) palette, and a set of semantic text colors.14

Each base color is expanded into a series of tokens, typically numbered from 10 (lightest) to 100 (darkest), allowing for fine-grained control over shades and tints. Key semantic colors are also given specific names for ease of use, such as Brand-primary for the main brand color (#006AF5) and text-primary for standard body text (#141415).15

The base color palettes include:

* Steel Blue
* Sky Blue
* Teal
* Blue
* Green
* Red
* Orange
* Yellow
* Pink
* Purple

The neutral palette consists of gray tones from gray-10 (#F7F7F8) to gray-100 (#0D0D0D).14

#### 4.2. Programmatic Usage of Color Tokens

To ensure consistency and maintainability, the color system is designed to be used programmatically through a dedicated NPM package, zaui-tokens. This allows developers to reference colors via variables rather than hard-coding hex values.14

**Installation:**

Bash

npm install zaui-tokens

CSS Usage:

The color tokens can be imported as CSS custom properties (variables) into a stylesheet. This is the recommended approach for styling components.

CSS

@import "zaui-tokens/variables.css";  
  
.my-component {  
 color: var(--colors-text-primary);  
 background-color: var(--colors-blue-10);  
}  
  
.my-primary-text {  
 color: var(--colors-blue-primary);  
}

14

JavaScript Usage:

The tokens can also be imported as a JavaScript object, which is useful for dynamic styling or use in component logic.

JavaScript

import { colors } from "zaui-tokens";  
  
// Example of accessing color values  
const primaryBlue = colors.blue.primary; // '#006AF5'  
const lightBlue = colors.blue; // '#F0F7FF'  
const defaultBlue = colors.blue.DEFAULT;  
  
console.log(primaryBlue);

14

### Chapter 5: Core Visual Primitives

Beyond typography and color, the ZAUI design system is defined by a set of core visual primitives that govern the layout and layering of components. These concepts are consistently referenced as foundational elements of the system.1

* **Corner Radius:** This primitive defines the curvature of corners on elements like buttons, inputs, and containers. Consistent application of corner radius values creates a cohesive and visually pleasing interface.
* **Spacing:** A systematic approach to spacing (margins and padding) ensures that layouts are balanced, breathable, and easy to scan. While specific token values are not detailed in the available documentation, the system's existence implies a predefined scale should be used.
* **Shadow:** Shadows are used to create a sense of elevation and depth, distinguishing interactive elements from the background and establishing a visual hierarchy.
* **Level Specification:** This refers to the standardized z-index layering model that governs how elements are stacked on the screen. It prevents visual conflicts and ensures that critical elements like modals and action sheets always appear above other content. The hierarchy is defined as follows:
  + **Level 1:** Main content of the page.
  + **Level 2:** Headers and navigation bars (Header, BottomNavigation).
  + **Level 3:** Overlays (Modal, Sheet).
  + Level 4: Action Sheets (ActionSheet).  
      
    16

## Part III: Architecture, Layout, and Navigation

This section details the high-level components and patterns used to structure an application's views, manage its overall layout, and handle navigation between different screens.

### Chapter 6: Core Container Components

#### 6.1. The App Component

The App component serves as the root container for a Zalo Mini App. It is the main component that wraps all other content and provides a centralized point for application-wide configuration and context.2

Any contexts or hooks provided at this level, such as theming, are made available to all descendant components. The primary documented property for the App component is theme, which allows for global theme customization.1

**Properties:**

| Name | Type | Default | Description |
| --- | --- | --- | --- |
| theme | "light" | "dark" | "light" | Customizes the theme for all ZAUI components within the App. |

Usage:

By wrapping the entire application in the App component, developers can easily switch between light and dark modes. Child components can then consume this theme value using the useTheme hook provided by zmp-ui to apply conditional styling.1

*Example (HomePage.js demonstrating theme consumption):*

JavaScript

import React from "react";  
import { Page, useTheme } from "zmp-ui";  
  
function HomePage() {  
 const [theme] = useTheme();  
  
 return (  
 <div  
 style={{  
 background: theme!== "dark"? "#FFFFFF" : "#141415",  
 color: theme!== "dark"? "#141415" : "#FFFFFF",  
 padding: "16px",  
 }}  
 >  
 <p>This component's background changes based on the theme.</p>  
 </div>  
 );  
}

2

#### 6.2. The Page Component

The Page component is a fundamental container designed to wrap the content of a single screen or view within the application.17 It provides essential functionality for managing the page's scroll behavior, which is critical for creating a smooth and professional user experience.

A significant architectural constraint was introduced in version 1.8.0 of the library. From this version forward, the Page component **must** be used in conjunction with a routing library, such as the standard react-router or the platform-provided ZMPRouter.17 This requirement enforces a Single-Page Application (SPA) architecture, where views are managed by a centralized router rather than through traditional multi-page browser navigation. This change mandates a more structured approach to application flow and view management.

**Properties:**

| Name | Type | Default | Description |
| --- | --- | --- | --- |
| children | ReactNode | - | The content to be rendered within the page. |
| hideScrollbar | boolean | false | Hides the page's scrollbar. |
| name | string | - | A unique name for the page, recommended when using scroll restoration features. |
| resetScroll | boolean | true | When using react-router, this scrolls the page to the top upon navigation. |
| restoreScroll | boolean | false | Always restores the scroll position when navigating back to this page. Ideal for tabbed interfaces. |
| restoreScrollOnBack | boolean | false | Restores the scroll position only when navigating back from a subsequent page. |

Usage:

The Page component is typically used as the top-level element for any component that represents a full screen. Its scroll restoration properties are particularly useful for complex user flows. For example, restoreScroll should be used for main tabs so that users do not lose their place when switching between them. For a detail page in a list, restoreScrollOnBack ensures that when the user returns to the list, they are at the same position they left.17

*Example (HomePage.js):*

JavaScript

import React from "react";  
import { Page, Input } from "zmp-ui";  
  
export default function HomePage(props) {  
 return (  
 <Page className="section-container" hideScrollbar={true}>  
 <Input label="Label" helperText="Helper text" />  
 {/\*... more page content... \*/}  
 </Page>  
 );  
}

17

### Chapter 7: Advanced Layout and Navigation

#### 7.1. Layout Primitives

To facilitate the creation of complex and responsive layouts, ZAUI provides a set of utility layout components. These primitives, introduced in version 1.10.0, offer declarative ways to arrange and align content, abstracting away common CSS patterns.19 The available components include:

* **Stack:** For arranging items in a vertical or horizontal line with consistent spacing.
* **Grid:** For creating grid-based layouts.
* **ZBox:** A general-purpose box model component, likely for custom layouts.
* **Center:** A component for centering its children both horizontally and vertically.
* **Cluster:** Likely used for grouping related items with consistent spacing.

1

#### 7.2. Header Component

The Header component provides the standard top navigation bar for a page. Its anatomy is fixed and consists of three main sections: a left area, typically for a back button; a central title; and a right-hand section containing the fixed Mini App controls (e.g., close, more options) provided by the Zalo platform.21

#### 7.3. BottomNavigation Component

The BottomNavigation component is the primary mechanism for top-level, persistent navigation within a Mini App. It is a bar fixed to the bottom of the screen, allowing users to switch between the main sections of the application.23

**Usage Guidelines:**

* It should contain between four and six navigation items.
* Each item must have both an icon and a label to clearly communicate its destination.
* It is intended for navigating between primary content destinations.

Anatomy:

The component is composed of a Container, Icon, Label, an optional Badge for notifications, and a Divider.24

**Properties:**

| Name | Type | Default | Description |
| --- | --- | --- | --- |
| fixed | boolean | false | If true, fixes the navigation bar to the bottom of the viewport. |
| activeKey | string | - | The key of the currently active item. This is a controlled property. |
| defaultActiveKey | string | - | The key of the item that should be active on initial render. |
| onChange | (activeKey: string) => void | - | Callback function that is invoked when the active item changes. |

Usage:

The component is used by nesting BottomNavigation.Item components within it. The activeKey and onChange props are used together to manage the component's state.

*Example:*

JavaScript

import React, { useState } from "react";  
import { BottomNavigation, Icon, Page } from "zmp-ui";  
  
const AppNavigation = () => {  
 const = useState("chat");  
  
 return (  
 <Page>  
 {/\* Page content for the active tab would be rendered here \*/}  
 <BottomNavigation  
 fixed  
 activeKey={activeTab}  
 onChange={(key) => setActiveTab(key)}  
 >  
 <BottomNavigation.Item  
 key="chat"  
 label="Tin Nhắn"  
 icon={<Icon icon="zi-chat" />}  
 activeIcon={<Icon icon="zi-chat-solid" />}  
 />  
 <BottomNavigation.Item  
 key="contact"  
 label="Danh bạ"  
 icon={<Icon icon="zi-call" />}  
 activeIcon={<Icon icon="zi-call-solid" />}  
 />  
 <BottomNavigation.Item  
 key="me"  
 label="Cá nhân"  
 icon={<Icon icon="zi-user" />}  
 activeIcon={<Icon icon="zi-user-solid" />}  
 />  
 </BottomNavigation>  
 </Page>  
 );  
};

24

#### 7.4. Tabs Component

The Tabs component is used for organizing and navigating between groups of content that are on the same hierarchical level within a single page. It is distinct from BottomNavigation, which is for app-level navigation.25

**Usage Guidelines:**

* Use for two or more fixed content sections.
* The content within each tab should be concise.

Anatomy:

The component consists of a Container, an Active Tab Item, Inactive Tab Items, and a Divider.25

**Properties:**

| Name | Type | Default | Description |
| --- | --- | --- | --- |
| scrollable | boolean | false | Allows horizontal scrolling if the tabs overflow the screen width. |
| activeKey | string | - | The key of the currently active tab. This is a controlled property. |
| defaultActiveKey | string | - | The key of the tab that should be active on initial render. |
| onChange | (activeKey: string) => void | - | Callback function invoked when the active tab changes. |

Usage:

The Tabs component can be used by nesting individual Tab components, each with a label and its corresponding content.

*Example:*

JavaScript

import { Tabs, Tab, Box, Text } from "zmp-ui";  
  
const ProductDetailsTabs = () => {  
 return (  
 <Tabs>  
 <Tab label="Giới thiệu">  
 <Box p={4}>  
 <Text>  
 Đây là nội dung giới thiệu sản phẩm.  
 </Text>  
 </Box>  
 </Tab>  
 <Tab label="Thông số">  
 <Box p={4}>  
 <Text>  
 Đây là các thông số kỹ thuật của sản phẩm.  
 </Text>  
 </Box>  
 </Tab>  
 </Tabs>  
 );  
};

25

## Part IV: Comprehensive Component Reference

This section provides a detailed reference for the individual UI components available in the ZAUI library. Each component is documented with its purpose, properties, usage guidelines, and code examples where available.

### Chapter 8: Data Display Components

#### 8.1. Avatar

**Purpose:** The Avatar component is used to display a user's profile image or a placeholder. It supports various states and can be grouped.

**Import:**

JavaScript

import { Avatar } from "zmp-ui";

**Properties (Avatar):**

| Name | Type | Default | Description |
| --- | --- | --- | --- |
| src | string | - | The URL for the avatar image. |
| size | number | - | The size of the avatar in pixels. |
| online | boolean | false | Displays a green dot indicating online status. |
| story | "default" | "seen" | - | Displays a border indicating a story is available (default) or has been viewed (seen). |
| blocked | boolean | false | Applies a visual treatment to indicate the user is blocked. |
| backgroundColor | string | - | Sets a gradient background. Accepts values like BLUE-BLUELIGHT, PURPLE-BLUE. |
| onClick | MouseEventHandler | - | Event handler for when the avatar is clicked. |
| children | React.ReactNode | - | Can be used to display text (e.g., initials) if src is not provided. |

26

**Properties (Avatar.Group):**

| Name | Type | Default | Description |
| --- | --- | --- | --- |
| maxCounter | number | - | The maximum number of avatars to display before showing a counter. |
| total | number | - | Overrides the count to display a specific total number. |
| horizontal | boolean | false | If true, arranges the avatars horizontally instead of stacked. |
| onCounterClick | MouseEventHandler | - | Event handler for when the overflow counter is clicked. |

26

Usage and Examples:

The Avatar component can be configured with various sizes and add-ons to convey user status. The Avatar.Group component is used to display a collection of avatars, such as members of a group chat.

*Example (Grouped Avatars):*

JavaScript

import { Avatar } from "zmp-ui";  
const { Group } = Avatar;  
  
const MemberList = () => {  
 return (  
 <Group maxCounter={3}>  
 <Avatar src="..." />  
 <Avatar src="..." />  
 <Avatar src="..." />  
 <Avatar src="..." />  
 </Group>  
 );  
};

26

#### 8.2. Icon

**Purpose:** The Icon component is used to render vector icons from the extensive Zalo Icon Kit.

**Import:**

JavaScript

import { Icon } from "zmp-ui";

Properties:

The available documentation does not specify the properties for the Icon component (e.g., size, color). However, it is typically used by passing an icon prop with the name of the desired icon.27

Available Icons:

The Zalo Icon Kit is organized into several categories, providing a comprehensive set of icons for common actions and objects. Icon names are prefixed with zi-.27

* **Arrow:** zi-arrow-down, zi-chevron-left, zi-download, etc.
* **Basic:** zi-check, zi-close, zi-plus, zi-home, zi-setting, etc.
* **Communication:** zi-chat, zi-call, zi-location, zi-send-solid, etc.
* **User:** zi-user, zi-group, zi-add-member, etc.
* **Signage:** zi-warning-solid, zi-info-circle, zi-help-circle, etc.

Usage:

The Icon component is frequently used within other components like Button, List.Item, and BottomNavigation.Item to provide visual cues.

*Example (Usage in a List Item):*

JavaScript

import { List, Icon } from "zmp-ui";  
const { Item } = List;  
  
const SettingsList = () => {  
 return (  
 <List>  
 <Item  
 title="Profile"  
 prefix={<Icon icon="zi-user" />}  
 suffix={<Icon icon="zi-chevron-right" />}  
 />  
 </List>  
 );  
};

28

#### 8.3. List

**Purpose:** The List component is used to display a vertically scrolling list of items. It supports dividers, loading states, and custom item rendering.

**Import:**

JavaScript

import { List } from "zmp-ui";  
const { Item } = List;

**Properties (List):**

| Name | Type | Default | Description |
| --- | --- | --- | --- |
| dataSource | array | - | An array of data to be rendered in the list. |
| renderItem | function | - | A function that takes an item from dataSource and returns a List.Item component. |
| loading | boolean | false | If true, displays a loading indicator. |
| divider | boolean | true | If true, displays a divider line between items. |
| children | React.ReactNode | - | Can be used to render static List.Items directly. |

**Properties (List.Item):**

| Name | Type | Default | Description |
| --- | --- | --- | --- |
| title | ReactNode | - | The main title of the list item. |
| subTitle | ReactNode | - | A secondary line of text displayed below the title. |
| prefix | ReactNode | - | A component (e.g., Avatar, Icon) to display at the beginning of the item. |
| suffix | ReactNode | - | A component (e.g., Icon, Switch) to display at the end of the item. |

28

Usage:

The List component can be used either declaratively with static Item children or dynamically by providing a dataSource and renderItem function.

*Example (Static List):*

JavaScript

import { List, Icon, Avatar } from "zmp-ui";  
const { Item } = List;  
  
const ContactList = () => {  
 return (  
 <List>  
 <Item  
 title="Đỗ Quang Minh"  
 subTitle="Quản lý dự án"  
 prefix={<Avatar online>M</Avatar>}  
 suffix={<Icon icon="zi-call" />}  
 />  
 <Item  
 title="Nguyễn Thị B"  
 subTitle="Nhà thiết kế"  
 prefix={<Avatar>B</Avatar>}  
 suffix={<Icon icon="zi-call" />}  
 />  
 </List>  
 );  
};

28

#### 8.4. Other Display Components

The ZAUI library includes several other components for data display, as indicated by the documentation structure.1 These include:

* **ImageViewer:** For displaying images in a fullscreen gallery or modal view.
* **Progress:** For indicating the progress of a task, such as a file upload or a multi-step process.
* **Spinner:** An indeterminate loading indicator for when the wait time is unknown.
* **Swiper:** A touch-enabled carousel for displaying a slideshow of images or content.
* **Text:** A component for rendering text with styles from the typography system.
* **Calendar:** A component for displaying dates and allowing date selection, added in version 1.11.0.19

### Chapter 9: Form and Input Components

#### 9.1. Button

**Purpose:** The Button component is used to trigger an action or event, such as submitting a form or opening a dialog.

**Import:**

JavaScript

import { Button } from "zmp-ui";

**Anatomy and Guidelines:**

* **Levels:** Buttons have three hierarchical levels to signify importance: Primary (for the main action), Secondary, and Tertiary (for less important actions).6
* **Sizes:** Three sizes are available: Large, Medium, and Small.
* **States:** Buttons have visual states for Pressed and Loading.
* **Content:** Buttons can contain text, an icon, or both. Icon-only buttons are typically used for supplementary actions.
* **Button Groups:** When grouping buttons, there should only be one Primary button to maintain a clear focal point. In horizontal groups, the primary button is on the right; in vertical groups, it is on top.6

Properties:

The documentation describes the anatomy and guidelines but does not provide a complete property table in the analyzed snippets.4 Common properties would include

size, level, loading, disabled, and onClick.

**Usage:**

JavaScript

import { Button } from "zmp-ui";  
  
const FormActions = () => {  
 return (  
 <div>  
 <Button level="primary" size="large">Submit</Button>  
 <Button level="tertiary" size="large">Cancel</Button>  
 </div>  
 );  
};

9

#### 9.2. Input

**Purpose:** The Input component is a versatile field for user text entry.

**Import:**

JavaScript

import { Input } from "zmp-ui";

**Properties:**

| Name | Type | Default | Description |
| --- | --- | --- | --- |
| type | string | "text" | The input type, e.g., "text", "password", "number". |
| label | string | - | A label displayed above the input field. |
| helperText | string | - | Informational text displayed below the input. |
| errorText | string | - | Error message displayed below the input when status is "error". |
| status | "error" | - | Sets the input to an error state, typically with a red border. |
| clearable | boolean | false | If true, shows a clear button when the input has value. |
| showCount | boolean | false | If true, displays a character count (requires maxLength). |
| maxLength | number | - | The maximum number of characters allowed. |
| disabled | boolean | false | Disables the input field. |
| prefix | ReactNode | - | A component to display at the beginning of the input. |
| suffix | ReactNode | - | A component to display at the end of the input. |
| value | string | - | The value of the input (controlled component). |
| onChange | (event) => void | - | Callback for when the input value changes. |

30

Usage:

The Input component can be configured for various use cases, from simple text entry to password fields with validation states.

*Example (Input with Error State):*

JavaScript

import React, { useState } from "react";  
import { Input } from "zmp-ui";  
  
const RegistrationForm = () => {  
 const [username, setUsername] = useState("");  
 const isError = username.length > 0 && username.length < 3;  
  
 return (  
 <Input  
 label="Username"  
 value={username}  
 onChange={(e) => setUsername(e.target.value)}  
 status={isError? "error" : undefined}  
 errorText={isError? "Username must be at least 3 characters." : ""}  
 helperText="Enter your desired username."  
 />  
 );  
};

30

#### 9.3. Selection Controls

ZAUI provides a standard set of components for making selections from a list of options.4

* **Checkbox:** Allows for the selection of zero, one, or multiple options from a set. A bug fix related to long labels was implemented in version 1.7.3.19
* **Radio:** Used when a user must select exactly one option from a set. Like Checkbox, it also received a fix for long labels in version 1.7.3.19
* **Select:** A dropdown menu for selecting one option from a list.
* **Picker:** A mobile-native picker interface (often a spinning wheel) for selecting values.
* **DatePicker:** A specialized picker for selecting dates and times.

#### 9.4. Switch

**Purpose:** The Switch component provides a toggle for binary (on/off) states.

**Import:**

JavaScript

import { Switch } from "zmp-ui";

**Properties:**

| Name | Type | Default | Description |
| --- | --- | --- | --- |
| label | string | - | A label displayed next to the switch. |
| size | "medium" | "small" | "medium" | The size of the switch component. |
| checked | boolean | - | The state of the switch (controlled component). |
| onChange | (checked: boolean) => void | - | Callback for when the switch state changes. |

31

#### 9.5. Slider

**Purpose:** The Slider component allows users to select a value from a continuous or discrete range by dragging a handle along a track.4

### Chapter 10: Overlay and Notification Components

#### 10.1. Modal

**Purpose:** The Modal component displays content in a layer that overlays the main application. It is used for critical information, user confirmation, or tasks that must be completed before returning to the main flow.

**Import:**

JavaScript

import { Modal } from "zmp-ui";

Anatomy:

A modal is composed of a Container, a Dim Background to obscure the page content, a Content View (which can be custom), and an Action area for buttons.20

**Behavior:**

* By default, modals can be dismissed by tapping the background (tapOutsideToDismiss).
* They can also be dismissed via an explicit dismissive button.
* Content within a modal can be scrollable if it exceeds the available space.20

Properties:

The available documentation describes the anatomy and behavior but does not provide a complete property table.20 Key properties would include

visible, onClose, title, and actions.

Usage:

The visibility of a modal is controlled by a state variable.

*Example:*

JavaScript

import React, { useState } from "react";  
import { Page, Button, Modal, Text } from "zmp-ui";  
  
const ModalExample = () => {  
 const [modalVisible, setModalVisible] = useState(false);  
  
 return (  
 <Page>  
 <Button onClick={() => setModalVisible(true)}>  
 Show Modal  
 </Button>  
 <Modal  
 visible={modalVisible}  
 title="Confirmation"  
 onClose={() => setModalVisible(false)}  
 actions={[  
 {  
 text: "Cancel",  
 onClick: () => setModalVisible(false),  
 },  
 {  
 text: "OK",  
 highLight: true,  
 onClick: () => setModalVisible(false),  
 },  
 ]}  
 >  
 <Text>Are you sure you want to proceed?</Text>  
 </Modal>  
 </Page>  
 );  
};

20

#### 10.2. Sheet

**Purpose:** The Sheet (or Bottom Sheet) is a component that slides up from the bottom of the screen to display supplementary content or a list of actions.

**Import:**

JavaScript

import { Sheet } from "zmp-ui";

Behavior:

Sheets can have multiple states or "snap points," including an initial height, an expanded (maximum) height, and a collapsed (minimum) height. The behavior can be configured to be dismissible or persistent.32

Properties:

The documentation describes behavior and provides a code example but does not include a full property table.32 Key properties would include

visible, onClose, and props to define the snap points.

Usage:

Similar to a modal, a sheet's visibility is controlled by component state.

*Example:*

JavaScript

import React, { useState } from "react";  
import { Page, Button, Sheet, Text } from "zmp-ui";  
  
const SheetExample = () => {  
 const = useState(false);  
  
 return (  
 <Page>  
 <Button onClick={() => setSheetVisible(true)}>  
 Show Sheet  
 </Button>  
 <Sheet  
 visible={sheetVisible}  
 onClose={() => setSheetVisible(false)}  
 autoHeight  
 >  
 <Text>This is the content of the bottom sheet.</Text>  
 </Sheet>  
 </Page>  
 );  
};

32

#### 10.3. ActionSheet

**Purpose:** The ActionSheet is a specific type of overlay, likely a variant of the Sheet, designed to present a user with a set of choices related to a task they initiated. It occupies the highest visual layer in the application, Level 4 of the Level Specification.16

#### 10.4. SnackbarProvider and useSnackbar

**Purpose:** The SnackbarProvider is a context provider that enables the display of "snackbars"—brief, non-intrusive messages that appear temporarily at the bottom of the screen to provide feedback on an operation.

Usage:

To use the snackbar functionality, the application must be wrapped with SnackbarProvider. Then, any child component can use the useSnackbar hook to programmatically trigger snackbars.4

Hook (useSnackbar):

The useSnackbar hook, provided by zmp-ui, returns an object with methods to control the snackbar, most notably openSnackbar.33

*Example:*

JavaScript

import { Page, Button, SnackbarProvider, useSnackbar } from "zmp-ui";  
  
// In a component that can trigger a snackbar  
const MyComponent = () => {  
 const { openSnackbar } = useSnackbar();  
  
 const showNotification = () => {  
 openSnackbar({  
 text: "Profile saved successfully!",  
 type: "success",  
 });  
 };  
  
 return <Button onClick={showNotification}>Save Profile</Button>;  
};  
  
// In the root of the application  
const App = () => {  
 return (  
 <SnackbarProvider>  
 <Page>  
 <MyComponent />  
 </Page>  
 </SnackbarProvider>  
 );  
};

33

## Part V: Advanced Topics and Ecosystem Integration

This final section explores the programmatic aspects of the ZAUI library, including its provided hooks, and examines how ZAUI components serve as the user-facing interface for the powerful Zalo Platform API.

### Chapter 11: Hooks and Utilities

#### 11.1. Provided Hooks

The zmp-ui package includes several React Hooks that provide access to component contexts and framework functionality, enabling more dynamic and interactive applications.

* **useTheme():** This hook provides access to the current theme ("light" or "dark") set by the root App component. It allows components to adapt their styles based on the global theme setting.1
* **useSnackbar():** This hook is used in conjunction with SnackbarProvider to programmatically open and manage snackbar notifications. It returns an object with methods like openSnackbar.33
* **useNavigate():** A hook for programmatic navigation. It is part of the ZMPRouter system (and likely a wrapper around react-router's hook) and is used to change pages in response to events that are not direct user clicks on a link.34

#### 11.2. Utility Functions

The official Zalo Mini App templates, such as zaui-coffee and zaui-market, consistently promote the practice of abstracting reusable, non-UI logic into a src/utils directory.5 This pattern is a recommended best practice for maintaining clean and testable code.

The purpose of this directory is to house pure functions that perform specific tasks, such as:

* **Data Formatting:** Functions for formatting dates, times, and currency values.
* **Calculations:** Business logic calculations, such as distance between two geographical points.
* **API Integration:** Helper functions for making API calls, handling authentication headers, and parsing responses.
* **Client-Side Logic:** Functions for managing complex client-side state or operations, such as cart management in an e-commerce app.

By separating this logic from the component rendering code, applications become more modular and easier to reason about.

### Chapter 12: Interacting with the Zalo Platform API (ZMP-SDK)

The true power of a Zalo Mini App is realized when the ZAUI component library is used in tandem with the Zalo Platform API (ZMP-SDK). The ZAUI components provide the interactive surface, while the ZMP-SDK provides access to the rich features of the Zalo platform and the user's device.3 The UI components are, in effect, the front-end for the SDK's capabilities.

The ZMP-SDK offers a wide range of APIs, including but not limited to:

* **User Information:** getUserInfo(), getPhoneNumber(), getAccessToken()
* **Device Access:** getLocation(), scanQRCode(), chooseImage()
* **Zalo Features:** openShareSheet(), followOA(), openChat()
* **UI Control:** showToast(), setNavigationBarTitle()

The standard development pattern involves attaching ZMP-SDK function calls to the event handlers of ZAUI components. The following table illustrates this synergistic relationship.

| Use Case | Relevant ZAUI Component(s) | ZMP-SDK API Call | Example Implementation |
| --- | --- | --- | --- |
| Request user's phone number | Button | getPhoneNumber() | A <Button onClick={() => getPhoneNumber({...})}> to initiate the permission request. |
| Share content via Zalo | Button, Icon | openShareSheet() | An <Icon icon="zi-share" onClick={() => openShareSheet({...})}> to open the native share dialog. |
| Scan a QR code | Button | scanQRCode() | A <Button onClick={() => scanQRCode({...})}> that opens the device camera for scanning. |
| Follow the brand's Official Account | Button | followOA() | A <Button onClick={() => followOA({...})}> to prompt the user to follow the linked OA. |
| Get the user's current location | Button | getLocation() | A <Button onClick={() => getLocation({...})}> to request location permissions and retrieve coordinates. |
| Show a temporary notification | Any component with an event | showToast() | An onClick handler that calls showToast({ message: "Success!" }) after an action. |

This integration model demonstrates that a comprehensive understanding of both the ZAUI component library and the ZMP-SDK is necessary to build fully-featured Zalo Mini Apps. The UI provides the means of interaction, while the SDK provides the powerful functionality that makes the platform compelling.

### Appendix: Version History

The release notes provide a valuable history of the ZAUI library's evolution, highlighting the introduction of new components and important bug fixes.19

* **Version 1.11.0 (15-04-2024):**
  + **New Features:** Introduced the Calendar component and added support for CSS variables for theming.
* **Version 1.10.0 (10-04-2024):**
  + **New Features:** Added a suite of layout components: ZBox, Center, Cluster, Grid, and Stack.
* **Version 1.9.5 (28-03-2024):**
  + **Bug Fixes:** Resolved an issue with the endDate prop on the DatePicker component.
* **Version 1.8.3 (15-11-2023):**
  + **Bug Fixes:** Addressed an issue where a disabled DatePicker was still selectable.
* **Version 1.7.3 (26-06-2023):**
  + **Bug Fixes:** Fixed an issue where long labels for Radio buttons and Checkboxes would not render correctly.
* **Older Versions:**
  + New components like ImageViewer and Swiper were released.
  + Fixes were implemented for input focus, Swiper active indicators, and modal positioning on older iOS versions.

#### Works cited

1. App | Zalo Mini App, accessed September 29, 2025, <https://mini.zalo.me/documents/zaui/container/App/>
2. App | Zalo Mini App, accessed September 29, 2025, <https://mini.zalo.me/documents/zaui/1.7.0/container/App/>
3. Giới Thiệu - Zalo Mini App, accessed September 29, 2025, <https://mini.zalo.me/documents/api/>
4. Introduction | Zalo Mini App, accessed September 29, 2025, <https://mini.zalo.me/documents/zaui/>
5. Zalo-MiniApp/zaui-coffee: Public template for building a ... - GitHub, accessed September 29, 2025, <https://github.com/Zalo-MiniApp/zaui-coffee>
6. Button | Zalo Mini App, accessed September 29, 2025, <https://mini.zalo.me/documents/zaui/1.9.0/button/>
7. Zalo Mini App Design Guidelines, accessed September 29, 2025, <https://mini.zalo.me/documents/intro/zalo-mini-app-design-guidelines/>
8. zmp-ui - Yarn Classic, accessed September 29, 2025, <https://classic.yarnpkg.com/en/package/zmp-ui>
9. Installation | Zalo Mini App, accessed September 29, 2025, <https://mini.zalo.me/documents/zaui/installation/>
10. zmp-ui - NPM, accessed September 29, 2025, <https://www.npmjs.com/package/zmp-ui/v/1.8.3?activeTab=code>
11. Installation | Zalo Mini App, accessed September 29, 2025, <https://stc-zmp.zdn.vn/zmp-docs/v3.10.0-sync-dts-to-docs.1/zaui/installation/>
12. Zalo-MiniApp/zaui-shop: E-commerce template for building Zalo mini program - GitHub, accessed September 29, 2025, <https://github.com/Zalo-MiniApp/zaui-shop>
13. Typography | Zalo Mini App, accessed September 29, 2025, <https://mini.zalo.me/documents/zaui/1.10.0/foundation/typography/>
14. Colors | Zalo Mini App, accessed September 29, 2025, <https://mini.zalo.me/documents/zaui/foundation/colors/>
15. Colors | Zalo Mini App, accessed September 29, 2025, <https://mini.zalo.me/documents/zaui/1.10.0/foundation/colors/>
16. Level Specification - Zalo Mini App, accessed September 29, 2025, <https://mini.zalo.me/documents/zaui/foundation/level-specification/>
17. Page | Zalo Mini App, accessed September 29, 2025, <https://mini.zalo.me/documents/zaui/container/Page/>
18. Page | Zalo Mini App, accessed September 29, 2025, <https://mini.zalo.me/documents/zaui/1.7.0/container/Page/>
19. Release Notes - Zalo Mini App, accessed September 29, 2025, <https://mini.zalo.me/documents/zaui/release-notes/>
20. Modal | Zalo Mini App, accessed September 29, 2025, <https://mini.zalo.me/documents/zaui/modal/>
21. Header | Zalo Mini App, accessed September 29, 2025, <https://mini.zalo.me/documents/zaui/layout/header/>
22. Header | Zalo Mini App, accessed September 29, 2025, <https://mini.zalo.me/documents/zaui/navigation/header/>
23. BottomNavigation | Zalo Mini App, accessed September 29, 2025, <https://mini.zalo.me/documents/zaui/layout/bottom-navigation/>
24. BottomNavigation | Zalo Mini App, accessed September 29, 2025, <https://mini.zalo.me/documents/zaui/navigation/bottom-navigation/>
25. Tabs | Zalo Mini App, accessed September 29, 2025, <https://mini.zalo.me/documents/zaui/layout/tabs/>
26. Avatar | Zalo Mini App, accessed September 29, 2025, <https://mini.zalo.me/documents/zaui/1.8.0/data-display/Avatar/>
27. Icons | Zalo Mini App, accessed September 29, 2025, <https://mini.zalo.me/documents/zaui/foundation/icons/>
28. List | Zalo Mini App, accessed September 29, 2025, <https://mini.zalo.me/documents/zaui/data-display/List/>
29. List | Zalo Mini App, accessed September 29, 2025, <https://mini.zalo.me/documents/zaui/display/list/>
30. Input | Zalo Mini App, accessed September 29, 2025, <https://mini.zalo.me/documents/zaui/form/input/>
31. Switch | Zalo Mini App, accessed September 29, 2025, <https://mini.zalo.me/documents/zaui/form/switch/>
32. Sheet | Zalo Mini App, accessed September 29, 2025, <https://mini.zalo.me/documents/zaui/sheet/>
33. Snackbar | Zalo Mini App, accessed September 29, 2025, <https://stc-zmp.zadn.vn/zmp-docs/v2.11.3/zaui/Snackbar/>
34. src/hooks.ts · main · ZAP / ZaloMiniApp · GitLab, accessed September 29, 2025, <https://gitlab.two.vn/zap/ZaloMiniApp/-/blob/main/src/hooks.ts>
35. Zalo-MiniApp/zaui-market - GitHub, accessed September 29, 2025, <https://github.com/Zalo-MiniApp/zaui-farmer-market>