ShadowUI

Indice generale

ShadowUI

Ottieni il massimo dal template per applicazioni moderne realizzate con Instant Developer Cloud

Introduzione

Nell'ambito delle applicazioni di trasformazione digitale, una delle necessità più frequenti è quella di sviluppare applicazioni moderne e professionali con un design system coerente e accessibile. Per ottenere questo risultato serve un framework che fornisca componenti altamente personalizzabili, aderenti alle linee guida moderne del web design e ottimizzati per ogni tipo di dispositivo.

Il framework deve generare un aspetto grafico professionale basato su design system consolidati come shadcn/ui e Radix, garantendo accessibilità (ARIA), temi personalizzabili (light/dark mode), e componenti responsive pronti all'uso.

Inoltre, deve tenere conto delle modalità di input moderne: tastiera, mouse, touch, dimensioni adattive per i controlli, transizioni fluide e feedback visivi immediati.
Il tutto senza dimenticarsi dell'utilizzo desktop tramite mouse e tastiera, che richiede ancora una volta un adattamento del contenuto al tipo di esperienza utente atteso per le applicazioni professionali.

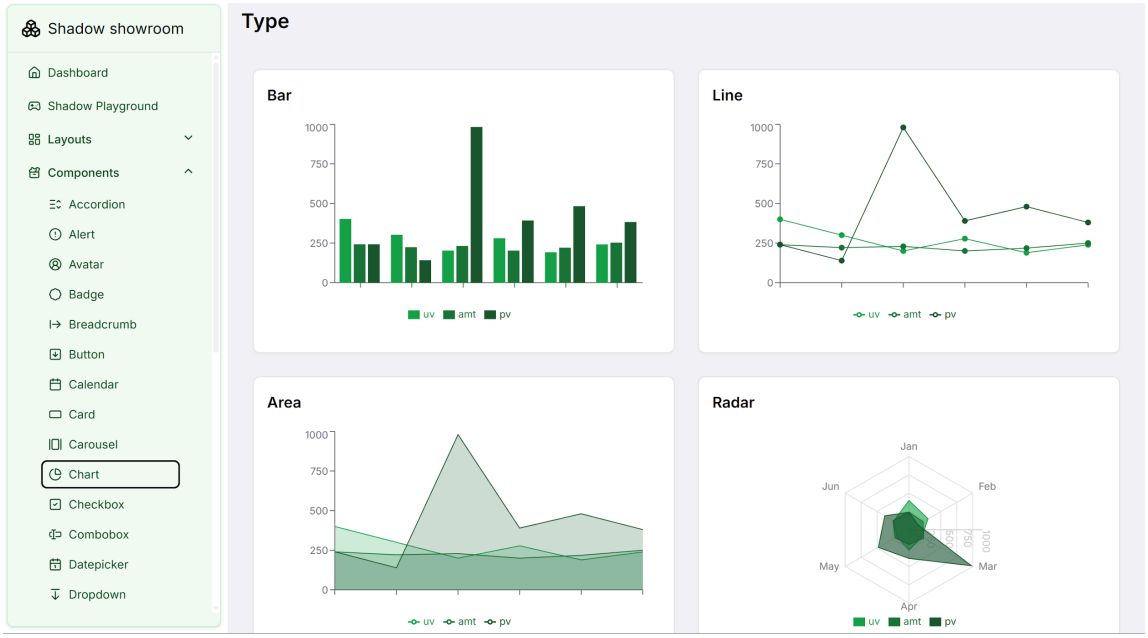
Per poter soddisfare queste necessità, Instant Developer Cloud contiene un set di elementi grafici denominato ShadowUI che permettono di realizzare applicazioni moderne e professionali in grado di funzionare correttamente su desktop, smartphone e tablet, con un design system coerente e personalizzabile.

ShadowUI è basato sui principi di shadcn/ui e Radix, utilizzando componenti altamente accessibili e personalizzabili tramite CSS Variables e Tailwind CSS.

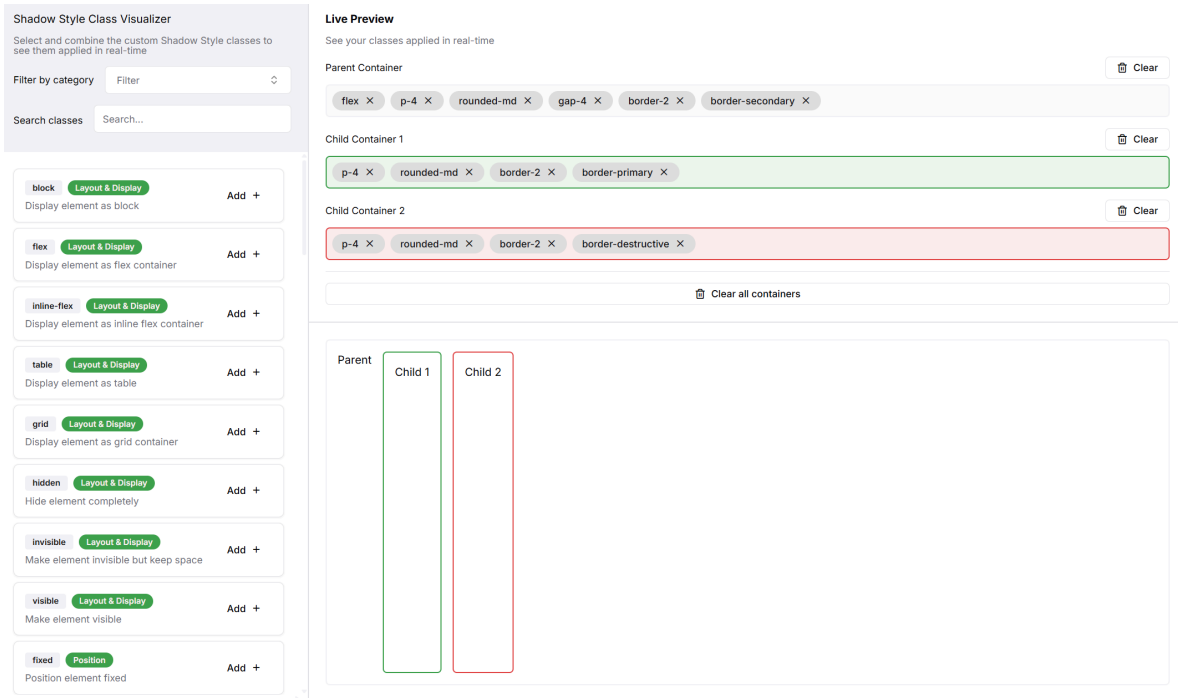
Progetti di esempio

Nella Console di Instant Developer Cloud sono presenti due progetti di esempio che aiutano nella comprensione e utilizzo del pacchetto *ShadowUI*.

Il progetto [ShadowUI Design Patterns] contiene esempi di utilizzo di tutti i 60+ componenti, i pattern di implementazione, le best practices corredati di codice completo.

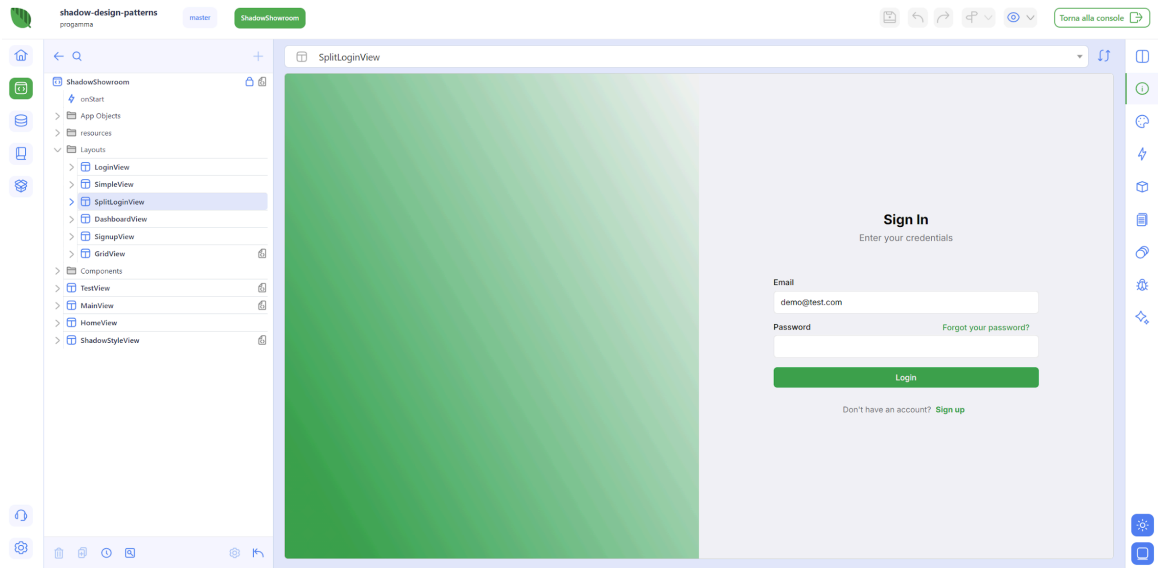


In questo progetto di esempio, nell'applicazione *Shadow showroom*, è presente la videata *Dashboard* dove sono elencati i principali pattern del componente. La videata *Shadow Playground* permette di prendere pratica con le classi CSS utilizzate dal framework.

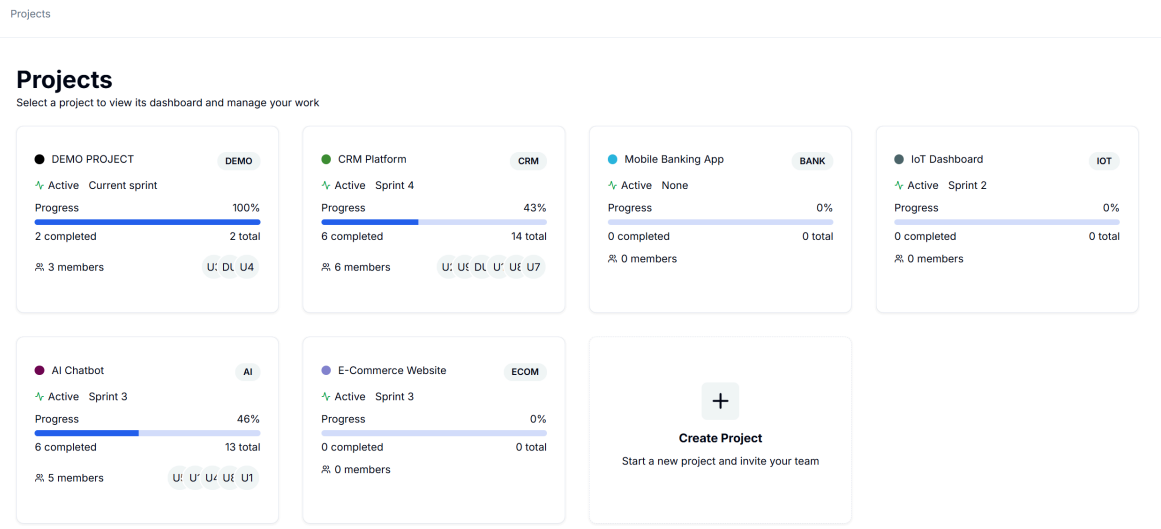


Qui è possibile vedere come le classi CSS *Tailwind* applicate ad una struttura predefinita ne modificano l'aspetto. Con questo tool si prende pratica nell'utilizzo delle classi CSS già presenti nel componente *ShadowUI*.

All'interno del progetto troviamo utilizzati tutti gli elementi e sono anche presenti alcuni esempi di utilizzo combinato che realizzano una semplice login, una login con immagine a destra, una dashboard, una videata di registrazione utente, una griglia di card.



Il progetto Sprinty è un'applicazione completa di project management con navigazione complessa, menu dinamico con DataMap, dashboard con grafici, form avanzati, gestione team e sprint.



Elementi disponibili

Gli elementi visuali disponibili in *ShadowUI* sono i seguenti:

- **struttura dell'applicazione**: ShaMainView, ShaMainViewSticky, ShaSidebarContainer, ShaNavController
- **struttura della pagina**: ShaPage, ShaHeaderBar, ShaSidebar, ShaSidebarHeader, ShaSidebarContent, ShaSidebarFooter, ShaSidebarPageContainer

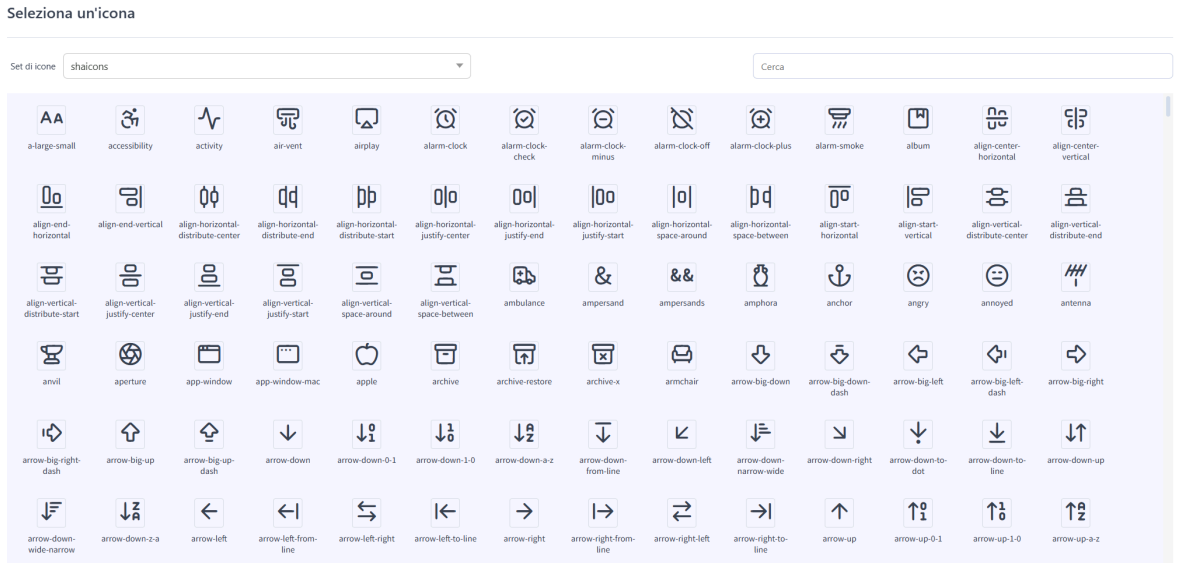
- **sidebar e menu**: ShaSidebarItem, ShaSidebarGroup, ShaMenubar, ShaMenubarTrigger, ShaMenubarItem, ShaMenubarCheckboxItem, ShaMenubarRadioGroup, ShaMenubarRadioItem, ShaMenubarSeparator
- **navigazione**: ShaBreadcrumb, ShaTabs, ShaTab
- **griglie e card**: ShaGrid, ShaRow, ShaCol, ShaCard, ShaCardHeader, ShaCardContent, ShaCardFooter
- **tabelle**: ShaTableElement (supporta table, tableHead, tableBody, tableFoot, tableRow, tableData, tableHeader, caption)
- **testi, label e icone**: ShaLabel, ShaBadge, ShaAlertBadge, ShaIcon, ShaAvatar
- **controlli input**: ShaButton, ShaInput, ShaCheckbox, ShaToggle, ShaSwitch, ShaRadioGroup, ShaRadioItem, ShaSelect, ShaCombobox, ShaDatePicker, ShaSlider, ShaInputOTP
- **layout e contenitori**: ShaSeparator, ShaAccordion, ShaCarousel, ShaProgress, ShaSkeleton
- **dropdown menu**: ShaDropdownMenu, ShaDropdownMenuItem, ShaDropdownMenuCheckboxItem, ShaDropdownMenuRadioGroup, ShaDropdownMenuRadioItem, ShaDropdownMenuLabel, ShaDropdownMenuSeparator, ShaDropdownMenuSub
- **widget avanzati**: ShaChart, ShaCalendar

Oltre a questi elementi è possibile utilizzare ogni altro elemento visuale disponibile, sia di tipo base che componenti finiti come mappe e grafici, e anche elementi personalizzati.

Nota bene: non è possibile aggiungere nella stessa applicazione altri framework grafici come ad esempio *IonicUI* o *Bootstrap*.

ShadowUI utilizza il set di icone Lucide Icons per default, un'evoluzione moderna di *Feather Icons*. È possibile aggiungere icon set personalizzati come verrà spiegato nei paragrafi successivi.

Le icone disponibili sono selezionabili visivamente mediante apposito dialog box:



Installazione

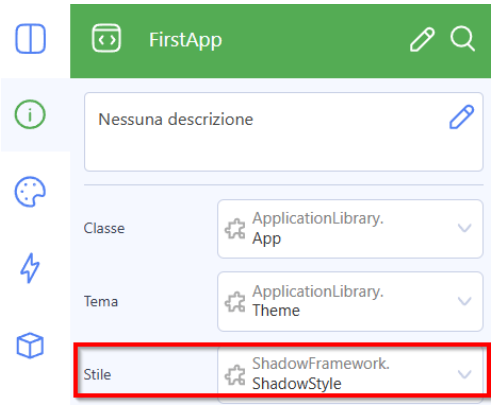
Per rendere disponibile *ShadowUI* in un proprio progetto è necessario importare il package ShadowUI tramite la videata apposita dell'IDE:

1. Aprire la sezione *Pacchetti del progetto*
2. Cliccare sul pulsante "+" per aggiungere un nuovo pacchetto
3. Selezionare *ShadowUI* dalla lista dei pacchetti disponibili
4. Cliccare su *Installa*

Una volta installato, tutti i componenti *ShadowUI* saranno disponibili nella palette degli elementi.
Nella sezione librerie vengono aggiunte:

- ShadowFramework - contiene tutti gli oggetti grafici del framework
- Shadow - contiene i page controller e alcuni template di oggetti nella videata *Template*

Per il corretto utilizzo delle classi Tailwind CSS occorre modificare la proprietà Style dell'applicazione impostando a a *ShadowFramwork.ShadowStyle*.



In questo modo quando si cerca una classe css nel box *Aggiungi classe CSS* della sezione *Stili* di un elemento a video saranno elencate anche le classi *Tailwind* di *ShadowUI*.

Le pagine ShadowUI

Il template *ShadowUI* permette di suddividere l'interfaccia utente dell'applicazione in pagine diverse, in cui ogni pagina corrisponde ad una videata. A differenza di *IonicUI*, *ShadowUI* offre due varianti di page controller, ottimizzate per diversi scenari d'uso.

ShaMainView vs ShaMainViewSticky

Vediamo le differenze tra i due page controller di *ShadowUI* confrontando le loro caratteristiche.

ShaMainView

Page controller base per applicazioni con sidebar standard.

Adatto per:

- Applicazioni con menu laterale sempre visibile
- Layout classico sidebar + contenuto
- Navigazione semplice senza breadcrumb permanente.

ShaMainViewSticky

Page controller avanzato con header sticky e breadcrumb.
Adatto per:

- Applicazioni professionali con navigazione complessa
- Header fisso in alto con breadcrumb navigation
- Sidebar collassabile con layout moderno
- Gestione automatica della cronologia di navigazione.

Nota importante: in questo manuale ci concentreremo su *ShaMainViewSticky* essendo la variante più completa e moderna.

Struttura del Page Controller

La struttura base di un page controller *ShadowUI* è costituita dai seguenti livelli.

Livello 1 - ShaSidebarContainer

- Contenitore principale con layout verticale
- Gestisce la divisione tra header (fisso in alto) e contenuto principale
- Visibile su tutti i dispositivi

Livello 2a - ShaHeaderBar

- Header fisso in alto
- Contiene:
    - Pulsante menu: Per aprire/chiudere la sidebar su mobile
    - ShaBreadcrumb: Navigazione automatica che mostra il percorso
    - ShaDropdownMenu: Menu utente con profilo, impostazioni, logout
- Sempre visibile durante lo scroll del contenuto

Livello 2b - mainCtn (Container)

- Contenitore orizzontale
- Divide lo spazio tra sidebar (a sinistra) e contenuto pagine (a destra)
- Si adatta automaticamente alla larghezza disponibile

Livello 3 - Sidebar e Contenuto

- **ShaSidebar**: Menu laterale collassabile
- **ShaSidebarHeader**: Logo, titolo progetto, badge
- **ShaSidebarContent**: Voci di menu (ShaSidebarGroup + ShaSidebarItem)
- **ShaSidebarFooter**: Impostazioni, info utente

- **ShaNavController**: Stack che contiene le pagine aperte, gestisce le transizioni tra pagine, mantiene lo stack di navigazione, supporta animazioni tra pagine

Struttura gerarchica completa

```
ShaMainViewSticky
└── ShaSidebarContainer
      ├── ShaHeaderBar (STICKY - fisso in alto)
      │     └── ShaBreadcrumb
      │
      └── mainCtn
            ├── ShaSidebar (laterale)
            │     ├── ShaSidebarHeader
            │     ├── ShaSidebarContent
            │     └── ShaSidebarFooter
            │
            └── ShaSidebarPageContainer
                  └── ShaNavController (gestisce stack pagine)
```

L'Intestazione (ShaHeaderBar)

L'header sticky contiene gli elementi fondamentali per la navigazione:

- **Pulsante menu**: Apre/chiude la sidebar (visibile su mobile)
- **Breadcrumb**: Mostra il percorso di navigazione corrente
- **Menu utente**: Dropdown con profilo, impostazioni, logout

Il breadcrumb viene aggiornato automaticamente dal page controller ogni volta che si naviga tra le pagine. Mostra il percorso completo di navigazione e permette di tornare a qualsiasi pagina precedente cliccando su di essa.
Il breadcrumb si popola in base al parametro title passato al metodo push():

```
view.push(App.DettaglioProgetto, {
  title: "Dettaglio Progetto", // ← Questo appare nel breadcrumb
  root: false
});
```

La Sidebar (ShaSidebar)

ShaSidebarHeader

L'header contiene tipicamente:

- Logo dell'applicazione
- Nome del progetto corrente
- Badge informativi (es. chiave progetto, stato)

ShaSidebarContent

Nel content vanno inseriti i menu di navigazione:

- **ShaSidebarGroup**: gruppi di menu con etichetta (es. "Navigation", "Admin")

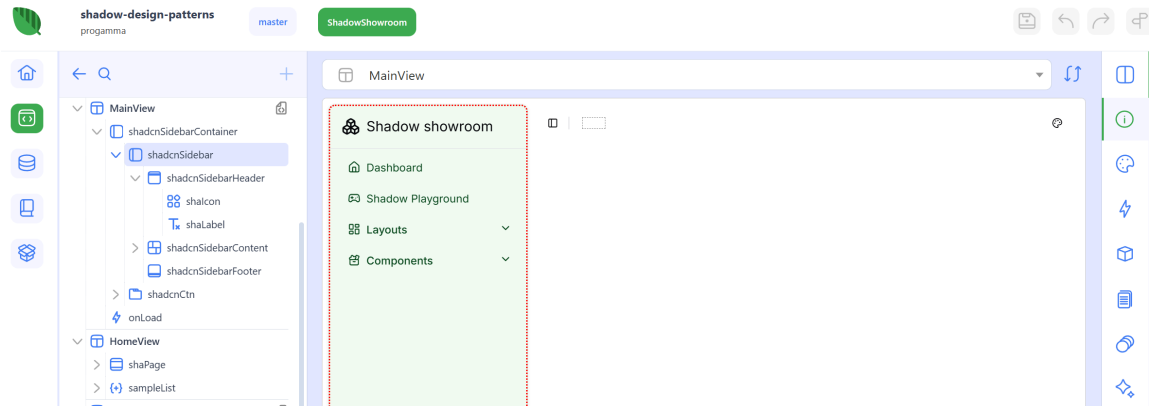
- **ShaSidebarItem**: singole voci di menu cliccabili

ShaSidebarFooter

Il footer contiene la sezione in fondo alla sidebar per elementi secondari:

- Impostazioni
- Help/Documentazione
- Info utente

Nell'immagine seguente un esempio di utilizzo di ShaMainView.



Il contenitore pagine (ShaNavController)

Il **ShaNavController** è il componente che gestisce lo stack di pagine aperte. Quando si chiamano i metodi push() o pop() del page controller, le videate vengono aggiunte o rimosse da questo stack.
Il nav controller gestisce automaticamente:

- Animazioni di transizione tra pagine
- Stack di navigazione (history)
- Evento onBack quando si torna indietro
- Rimozione delle pagine dallo stack

Navigazione: proprietà e metodi del Page Controller

Sia **ShaMainView** che **ShaMainViewSticky** ereditano una serie di proprietà e metodi per gestire la navigazione tra le pagine dell'applicazione.
Questi metodi permettono di:

- Aprire nuove pagine (push)
- Tornare indietro (pop)
- Rimuovere pagine dallo stack (remove)
- Controllare la sidebar (toggleMenu)
- Aggiornare breadcrumb e titoli (setTitle, setBreadcrumb).

Proprietà principali

Proprietà	Tipo	Descrizione
navStack	array	Stack delle view navigate (cronologia)
removing	boolean	Flag per evitare operazioni concorrenti durante rimozione
actualActivePageIndex	number	Indice della pagina attualmente attiva
menuDisabled	boolean	Stato disabilitato del menu sidebar
animateView	boolean	Abilita/disabilita animazioni di transizione

Il navStack contiene oggetti con questa struttura:

```
{
  view: viewInstance,       // Istanza della view
  options: { ... },         // Opzioni passate a push()
  index: 0                  // Posizione nel ShaNavController
}
```

Metodo push()

Naviga a una nuova view aggiungendola allo stack, questa la sua sintassi:

```
view.push(viewclass, options)
```

Parametri options:

Parametro	Tipo	Descrizione
root	boolean	Se true, sostituisce tutto lo stack (pagina principale)
popup	boolean	Apre come popup invece che nel contenitore
animate	boolean	Abilita animazione di transizione (default: true)
wait	boolean	Attende 10ms prima di eseguire la transizione
remove	boolean	Rimuove le view precedenti dallo stack
title	string	Titolo da mostrare nel breadcrumb

Esempi:

```
// 1. Navigazione standard
function* dashboardItem_onClick() {
  yield view.push(App.DashboardView, {
    title: "Dashboard"
  });
}
```

```
// 2. Apertura dettaglio da lista
function* ticketCard_onClick() {
  let ticket = this.sender.row;
  yield view.push(App.TicketDetailView, {
    title: `Ticket #${ticket.id}`,
    ticketId: ticket.id
  });
}

// 3. Pagina principale (root) - sostituisce tutto lo stack
function* projectsItem_onClick() {
  yield view.push(App.ProjectsView, {
    root: true,
    title: "Progetti"
  });
}


// 4. Apertura come popup
function* settingsButton_onClick() {
  yield view.push(App.SettingsView, {
    popup: true,
    title: "Impostazioni"
  });
}
```

Comportamento con root impostato a true:

- Elimina tutte le view precedenti dallo stack
- Disabilita l'animazione di default
- La nuova view diventa la "radice" della navigazione
- Non mostra pulsante "indietro" nel breadcrumb

Metodo pop()

Torna indietro nello stack, chiudendo le view successive.
Sintassi:

```
view.pop(viewOrCount, info)
```

Parametri:

- viewOrCount: Numero di view da chiudere oppure classe della view target
- info: Oggetto con dati da passare alla view precedente (opzionale)

Esempi:

```
// 1. Torna indietro di una view
function* backButton_onClick() {
  yield view.pop();
}

// 2. Torna indietro di 3 view
function* closeAllButton_onClick() {
  yield view.pop(3);
}

// 3. Torna a una view specifica
function* backToDashboard_onClick() {
  yield view.pop(App.DashboardView);
}


// 4. Torna indietro passando dati
function* saveButton_onClick() {
  yield this.view.ticketDM.save();

  yield view.pop(1, {
    saved: true,
    ticketId: this.ticket.id
  });
}
```

Per ricevere i dati con l'evento onBack:
La view precedente può intercettare il ritorno implementando l'evento onBack:

```
// In TicketListView
function* onBack(closedView, info) {
  if (info.saved) {
    // Ricarica la lista
    yield this.view.ticketsDM.load();

    yield app.toast(`Ticket ${info.ticketId} salvato`, {
      variant: "success"
    });
  }
}
```

Metodo remove()

Rimuove view dallo stack senza navigare indietro. Utile per pulire lo stack mantenendo la view corrente.
Sintassi:

```
view.remove(viewOrCount, info)
```

Parametri:

- viewOrCount: Numero di view da rimuovere oppure classe della view oppure true per rimuovere tutto
- info: Oggetto con informazioni (opzionale)

Esempi:

```
// 1. Rimuovi la view precedente (quella prima della corrente)
yield view.remove(1);

// 2. Rimuovi tutte le view tranne la corrente
yield view.remove(true);


// 3. Flusso: Dashboard → List → Detail → Edit
// Dopo save, vuoi: Dashboard → Detail (salta List)
function* saveButton_onClick() {
  yield this.saveProject();

  // Rimuovi List dallo stack
  yield view.remove(2);

  // Torna a Detail
  yield view.pop(1, { saved: true });
}
```

Metodo toggleMenu()

Apre o chiude la sidebar.
Sintassi:

```
view.toggleMenu(show)
```

Parametri:

- show (boolean): true per aprire, false per chiudere, undefined per toggle

Esempi:

```
// 1. Toggle (apri se chiusa, chiudi se aperta)
function* menuButton_onClick() {
  yield view.toggleMenu();
}


// 2. Chiudi menu dopo selezione su mobile
function* menuItem_onClick() {
  yield view.push(App.TargetView, { title: "Pagina" });

  if (app.device.isMobile) {
    yield view.toggleMenu(false);
  }
}
```

Metodo setTitle() - Solo ShaMainViewSticky

Aggiorna il titolo dell'ultima view nello stack e rigenera il breadcrumb.
Sintassi:

```
view.setTitle(newTitle)
```

Esempi:

```
// 1. Aggiorna titolo dopo caricamento dati
function* onLoad() {
  yield this.view.projectDM.load();

  let project = this.view.projectDM.getData();
  view.setTitle(`Progetto: ${project.name}`);
}

// 2. Aggiorna titolo durante modifica
function* titleInput_onChange() {
  let title = this.view.titleInput.value;
  view.setTitle(`Modifica: ${title}`);
}
```

Metodo setBreadcrumb()

Rigenera il breadcrumb basandosi sullo stack corrente. Viene chiamato automaticamente da push(), pop(), e setTitle().
Il breadcrumb mostra tutte le view con un title nelle options (esclusi i popup).

Navigazione breadcrumb: Quando l'utente clicca su un elemento del breadcrumb, viene automaticamente chiamato pop() per tornare a quella view.

Metodo getActivePage()

Ottiene la view attualmente attiva (l'ultima nello stack).
Sintassi:

```
let currentView = view.getActivePage();
```

Esempio:

```
function* refreshButton_onClick() {
  let activeView = view.getActivePage();

  if (activeView.refreshData) {
    yield activeView.refreshData();
  }
}
```

Metodo postMessage()

Invia un messaggio alle view nello stack. Di default invia solo all'ultima view, con bc: true invia a tutte (broadcast).
Sintassi:

```
view.postMessage(message)
```

Esempi:

```
// 1. Notifica aggiornamento dati alla view corrente
yield view.postMessage({
  type: "data-updated",
  entity: "ticket",
  id: 123
});

// In TicketListView - Ricevi messaggio
function* onMessage(message) {
  if (message.type === "data-updated") {
    yield this.view.ticketsDM.load();
  }
}

// 2. Broadcast a tutte le view
yield view.postMessage({
  bc: true,
  type: "theme-changed",
  theme: "dark"
});
```

Pattern comuni

Flusso Master-Detail.

```
// Lista → Dettaglio → Modifica → Torna a Dettaglio

// ProjectsView
function* projectCard_onClick() {
  let project = this.sender.row;

  yield view.push(App.ProjectDetailView, {
    title: project.name,
    projectId: project.id
  });
}
```

```
// ProjectDetailView
function* editButton_onClick() {
  yield view.push(App.ProjectEditView, {
    title: "Modifica Progetto",
    projectId: this.options.projectId
  });
}

// ProjectEditView
function* saveButton_onClick() {
  yield this.view.projectDM.save();
  yield view.pop(1, { saved: true });
}

// ProjectDetailView - onBack
function* onBack(closedView, info) {
  if (info.saved) {
    yield this.view.projectDM.load();
    yield app.toast("Salvato", { variant: "success" });
  }
}
```

Logout con pulizia stack.

```
function* logoutButton_onClick() {
  let confirm = yield app.popup(
    "Conferma",
    "Sei sicuro di voler uscire?",
    ["Annulla", "Esci"]
  );

  if (confirm === 1) {
    yield app.logout();

    // Torna a login pulendo tutto
    yield view.push(App.LoginView, {
      root: true,
      remove: true,
      animate: false
    });
  }
}
```

Menu dinamico con DataMap.

```
// MainPage - onLoad
function* onLoad() {
  yield this.view.projectsDM.load();
}
```

```
// MainPage - projectsDM_onRowComposition
function* projectsDM_onRowComposition(row) {
  let item = this.view.addChild(
    ShaSidebarItem,
    this.view.projectsGroup
  );

  item.label = row.name;
  item.icon = "shaicons shaicons-folder";
  item.row = row;
}

// MainPage - projectItem_onClick
function* projectItem_onClick() {
  let project = this.sender.row;

  yield view.push(App.ProjectDetailView, {
    root: true,
    title: project.name,
    projectId: project.id
  });
}
```

Differenze ShaMainView vs ShaMainViewSticky.

Caratteristica	ShaMainView	ShaMainViewSticky
Header	Scrolla con contenuto	Fisso in alto (sticky)
Breadcrumb	Dentro mainCtn	Nel livello top (sempre visibile)
Metodo setTitle	Non disponibile	Disponibile
Sidebar padding	Standard	padding-top: 4rem
Z-index header	Standard	11 (sempre sopra)
Caso d’uso	App semplici	App professionali con navigazione complessa

Best Practices

Titoli descrittivi

Usa sempre titoli significativi per il breadcrumb.

```
// ❌ Male
yield view.push(App.DetailView, { title: "Dettagli" });

// ✅ Bene
yield view.push(App.TicketDetailView, {
  title: `Ticket #${ticket.id} - ${ticket.title}`
});
```

Root per pagine principali

Usa root: true solo per view di primo livello.

```
// Dashboard, Projects, Settings = root
yield view.push(App.DashboardView, { root: true, title: "Dashboard" });

// Detail, Edit, etc = NO root
yield view.push(App.TicketDetailView, { title: "Ticket #123" });
```

Pulizia stack

Evita stack troppo profondi.

```
// Se lo stack diventa: A → B → C → D → E
// E non serve tornare a B, C, D:
yield view.remove(3); // Rimuovi B, C, D
yield view.pop(); // Torna ad A
```

Feedback visivo

Disabilita pulsanti durante la navigazione.

```
function* openButton_onClick() {
  this.view.openButton.disabled = true;

  yield view.push(App.TargetView, { title: "..." });

  this.view.openButton.disabled = false;
}
```

Creare un Page Controller nella propria applicazione

Per utilizzare il page controller nella propria applicazione, seguire questi passaggi:

1. Creare una videata, ad esempio di nome MainPage
2. Cambiare la classe della videata da *Application.View* a *Shadow.ShaMainViewSticky*
3. La struttura base apparirà automaticamente nella videata

Subito dopo aver impostato l'estensione, all'interno della videata che prima era vuota appariranno le strutture personalizzabili del page controller.

Il page controller è quindi costituito da: - Una sezione visuale per il menu (da personalizzare in ShaSidebarContent) - Una serie di metodi ereditati dalla classe base per gestire la navigazione (push, pop, toggleMenu, ecc.)

Definire le voci di menu

Per definire le voci di menu, inserire gli elementi all'interno di ShaSidebarContent:

1. Aprire la videata MainPage nell'IDE
2. Navigare a: shadcnSidebarContainer > mainCtn > shadcnSidebar > shadcnSidebarContent
3. Aggiungere ShaSidebarGroup per raggruppare le voci
4. All'interno di ogni gruppo, aggiungere ShaSidebarItem

5. Per ogni ShaSidebarItem, impostare:
   - label: Testo della voce
   - icon: Icona (es. “shaicons shaicons-home”)
   - iconPosition: Posizione icona (left/right)
6. Gestire l’evento onClick di ogni item per aprire la videata corrispondente

Esempio implementazione:

```
// MainPage - dashboardItem onClick
function* dashboardItem_onClick() {
  yield view.push(App.DashboardView, {
    root: true,
    title: "Dashboard"
  });

  // Chiudi menu su mobile
  if (app.device.isMobile) {
    yield view.toggleMenu(false);
  }
}
```

La HeaderBar e il Breadcrumb

L’header bar è il componente fisso in alto che contiene la navigazione e i controlli principali dell’applicazione.

ShaHeaderBar

La *ShaHeaderBar* è il contenitore dell’intestazione dell’applicazione. In *ShaMainViewSticky*, questo elemento rimane fisso in alto durante lo scroll della pagina.
Proprietà principali:

Proprietà	Tipo	Descrizione
sidebarButton	boolean	Mostra il pulsante per aprire la sidebar (default: true)
className	string	Classi CSS personalizzate
style	object	Stili inline personalizzati

Contenuto standard:

- Pulsante menu sidebar (su mobile)
- ShaBreadcrumb: Navigazione breadcrumb
- ShaDropdownMenu: Menu utente

ShaBreadcrumb

Il *breadcrumb* mostra il percorso di navigazione corrente e permette di tornare alle pagine precedenti.

Proprietà principali:

Proprietà	Tipo	Descrizione
items	array	Array di oggetti { id, title } da visualizzare
maxNames	number	Numero massimo di elementi da mostrare (default: 4)
disabled	boolean	Disabilita l'interazione
visible	boolean	Mostra/nascondi il breadcrumb

Struttura items:

```
[
  { id: 0, title: "Dashboard" },
  { id: 1, title: "Progetti" },
  { id: 2, title: "Progetto Alpha" }
]
```

Eventi:
onSelect(event):

- Scatenato quando si clicca su un elemento del breadcrumb
- event.id: id dell'elemento cliccato

Aggiornamento automatico:
Il breadcrumb viene aggiornato automaticamente quando si usano i metodi di navigazione (push, pop, setTitle). Non è necessario gestirlo manualmente.

Comportamento al click:
Quando l'utente clicca su un elemento del breadcrumb, il page controller chiama automaticamente pop() per tornare a quella pagina:

```
// Gestito automaticamente dal framework
shadcnBreadcrumb.onSelect = function(event) {
  let id = event.id;

  if (id >= (view.navStack.length - 1)) return;

  let nrPageToRemove = view.navStack.length - id - 1;
  view.pop(nrPageToRemove);
};
```

Esempio personalizzazione:

```
// Imposta breadcrumb manualmente (raramente necessario)
function* updateBreadcrumb() {
  this.view.shadcnBreadcrumb.items = [
    { id: 0, title: "Home" },
    { id: 1, title: "Progetti" },
    { id: 2, title: this.currentProject.name }
  ];
}
```

ShaDropdownMenu

Il dropdown menu nell'header contiene tipicamente le azioni utente: profilo, impostazioni, logout.
Struttura:

```
ShaDropdownMenu
├── ShaDropdownMenuItem (Profilo)
├── ShaDropdownMenuItem (Impostazioni)
├── ShaDropdownMenuSeparator
└── ShaDropdownMenuItem (Logout)
```

Di seguito analizziamo i componenti del dropdown.

ShaDropdownMenu

Contenitore principale del menu a tendina.
Proprietà:

- label: Testo del trigger button
- icon: Icona del trigger
- align: Allineamento menu (start, center, end)

ShaDropdownMenuItem

Voce di menu cliccabile.
Proprietà:

- label: Testo della voce
- icon: Icona (opzionale)
- disabled: Disabilita la voce

Eventi: - onSelect(event): Scatenato al click
Esempio:

```
// Dropdown menu utente
function* profileMenuItem_onSelect() {
  yield view.push(App.ProfileView, {
    title: "Profilo Utente"
  });
}
```

```
function* logoutMenuItem_onSelect() {
  let confirm = yield app.popup(
    "Conferma Logout",
    "Sei sicuro di voler uscire?",
    ["Annulla", "Esci"]
  );

  if (confirm === 1) {
    yield app.logout();
    yield view.push(App.LoginView, {
      root: true,
      remove: true
    });
  }
}
```

ShaDropdownMenuCheckboxItem

Voce con checkbox per opzioni on/off.
Proprietà:

- label: Testo
- checked: Stato checkbox (boolean)

Eventi:

- onCheckedChange(event): Scatenato al cambio stato

Esempio:

```
// Toggle notifiche
function* notificationsCheckbox_onCheckedChange() {
  let enabled = this.sender.checked;

  yield app.settings.setNotifications(enabled);

  yield app.toast(
    enabled ? "Notifiche attivate" : "Notifiche disattivate",
    { variant: "success" }
  );
}
```

ShaDropdownMenuRadioGroup e RadioItem

Per selezione esclusiva tra opzioni.
Esempio selezione tema:

```
ShaDropdownMenuRadioGroup
├── ShaDropdownMenuRadioItem (Light)
├── ShaDropdownMenuRadioItem (Dark)
└── ShaDropdownMenuRadioItem (Auto)
```

```
function* themeRadioItem_onSelect() {
  let theme = this.sender.value; // "light", "dark", "auto"

  yield app.theme.set(theme);

  // Notifica tutte le view aperte
  yield view.postMessage({
    bc: true,
    type: "theme-changed",
    theme: theme
  });
}
```

ShaDropdownMenuSeparator

Linea separatrice tra gruppi di voci.

ShaDropdownMenuLabel

Etichetta non cliccabile per raggruppare voci.
Esempio struttura completa:

```
ShaDropdownMenu
├── ShaDropdownMenuLabel ("Account")
├── ShaDropdownMenuItem ("Profilo")
├── ShaDropdownMenuItem ("Impostazioni")
├── ShaDropdownMenuSeparator
├── ShaDropdownMenuLabel ("Preferenze")
├── ShaDropdownMenuCheckboxItem ("Notifiche")
├── ShaDropdownMenuSeparator
└── ShaDropdownMenuItem ("Logout")
```

Pattern comuni HeaderBar

Menu utente con avatar

```
ShaDropdownMenu
  label: app.user.name
  icon: app.user.avatar
  ├── MenuItem (Profilo)
  ├── MenuItem (Impostazioni)
  ├── Separator
  └── MenuItem (Logout)
```

Breadcrumb con limite elementi

```
ShaBreadcrumb
  maxNames: 3  // Mostra max 3 elementi

// Se ci sono più elementi: "Home > ... > Dettaglio"
```

Aggiornamento dinamico breadcrumb

```
// Nel metodo onLoad della view
function* onLoad() {
  yield this.view.dataDM.load();

  let data = this.view.dataDM.getData();

  // Aggiorna titolo breadcrumb
  view.setTitle(`${data.type}: ${data.name}`);
}
```

Header con azioni contestuali

```
// Aggiungi pulsanti nell'header per azioni della pagina corrente
ShaHeaderBar
  └── actionsContainer
        ├── ShaButton (Salva)
        ├── ShaButton (Annulla)
        └── ShaButton (Elimina)
```

I pulsanti

I pulsanti sono elementi fondamentali per l'interazione utente. ShadowUI offre il componente *ShaButton* con diverse varianti e opzioni.

ShaButton

Proprietà principali:

Proprietà	Tipo	Descrizione
text	string	Testo del pulsante
variant	enum	Stile: default, secondary, outline, destructive, ghost, link
icon	string	Classe icona (es. "shaicons shaicons-save")
iconPosition	enum	Posizione icona: left, right, only
disabled	boolean	Stato disabilitato
loading	boolean	Mostra spinner di caricamento
size	enum	Dimensione: sm, default, lg
className	string	Classi CSS personalizzate

Eventi:

- onClick(event): Scatenato al click
- onMouseOver(event): Scatenato al passaggio del mouse
- onMouseOut(event): Scatenato quando il mouse esce

Varianti pulsante

1. Default

Pulsante primario con sfondo colorato.

```
ShaButton
  text: "Salva"
  variant: "default"
  icon: "shaicons shaicons-save"
  iconPosition: "left"
```

Uso: Azione principale della pagina (es. Salva, Conferma, Invia)

2. Secondary

Pulsante secondario con sfondo grigio.

```
ShaButton
  text: "Annulla"
  variant: "secondary"
```

Uso: Azioni secondarie (es. Annulla, Chiudi).

3. Outline

Pulsante con solo bordo colorato.

```
ShaButton
  text: "Modifica"
  variant: "outline"
  icon: "shaicons shaicons-edit"
```

Uso: Azioni terziarie, pulsanti in gruppi.

4. Destructive

Pulsante rosso per azioni distruttive.

```
ShaButton
  text: "Elimina"
  variant: "destructive"
  icon: "shaicons shaicons-trash-2"
```

Uso: Azioni pericolose (Elimina, Rimuovi, Cancella).

5. Ghost

Pulsante quasi trasparente, senza bordo.

```
ShaButton
  text: "Dettagli"
  variant: "ghost"
```

Uso: Azioni discrete, pulsanti in tabelle

6. Link

Pulsante con aspetto di link testuale.

```
ShaButton
  text: "Scopri di più"
  variant: "link"
```

Uso: Link interni, azioni secondarie inline

Posizioni icona

Icon Left

Icona a sinistra del testo.

```
ShaButton
  text: "Nuovo Progetto"
  icon: "shaicons shaicons-plus"
  iconPosition: "left"
```

Icon Right

Icona a destra del testo.

```
ShaButton
  text: "Esporta"
  icon: "shaicons shaicons-download"
  iconPosition: "right"
```

Icon Only

Solo icona, senza testo.

```
ShaButton
  icon: "shaicons shaicons-settings"
  iconPosition: "only"
  variant: "ghost"
```

Uso: Toolbar, azioni compatte.

Stati pulsante

Disabled

```
ShaButton
  text: "Salva"
  disabled: true
```

Loading

```
ShaButton
  text: "Caricamento..."
  loading: true
```

Esempio gestione loading:

```
function* saveButton_onClick() {
  // Attiva loading
  this.view.saveButton.disabled = true;
  this.view.saveButton.loading = true;
  this.view.saveButton.text = "Salvataggio...";

  try {
    yield this.view.dataDM.save();

    yield app.toast("Salvato con successo", {
      variant: "success"
    });
  } finally {
    // Disattiva loading
    this.view.saveButton.disabled = false;
    this.view.saveButton.loading = false;
    this.view.saveButton.text = "Salva";
  }
}
```

Dimensioni

```
// Small
ShaButton
  text: "Piccolo"
  size: "sm"

// Default
ShaButton
  text: "Normale"
  size: "default"

// Large
ShaButton
  text: "Grande"
  size: "lg"
```

Gruppi di pulsanti

Per azioni multiple correlate:

```
ShaRow
  className: "gap-2"
  ├── ShaButton (Salva - default)
  ├── ShaButton (Salva e continua - secondary)
  └── ShaButton (Annulla - outline)
```

Keyboard shortcuts

Puoi associare shortcut da tastiera ai pulsanti:

```
ShaButton
  text: "Salva"
  cmdKey: "s"  // Ctrl+S / Cmd+S
```

Altri esempi:

- cmdKey: "n" → Ctrl+N (Nuovo)
- cmdKey: "f" → Ctrl+F (Cerca)
- cmdKey: "p" → Ctrl+P (Stampa)

Pattern comuni

1. Pulsanti form:

```
ShaCardFooter
  ├── ShaButton
  │     text: "Salva"
  │     variant: "default"
  │     icon: "shaicons shaicons-save"
  │
  ├── ShaButton
  │     text: "Salva e continua"
  │     variant: "secondary"
  │
  └── ShaButton
        text: "Annulla"
        variant: "outline"
```

2. Toolbar azioni:

```
ShaRow
  className: "gap-1"
  ├── ShaButton (icon only - edit)
  ├── ShaButton (icon only - duplicate)
  ├── ShaButton (icon only - delete)
  └── ShaSeparator (vertical)
```

3. Pulsante con conferma:

```
function* deleteButton_onClick() {
  let confirm = yield app.popup(
    "Conferma eliminazione",
    "Questa operazione non può essere annullata.",
    ["Annulla", "Elimina"]
  );

  if (confirm === 1) {
    yield this.deleteItem();
  }
}
```

4. Pulsante disabilitato condizionalmente:

```
function* formInput_onChange() {
  // Disabilita salva se form non valido
  let isValid = this.validateForm();
  this.view.saveButton.disabled = !isValid;
}
```

5. Pulsante dropdown combo:

```
ShaRow
  ├── ShaButton (Salva)
  └── ShaDropdownMenu
        trigger: ShaButton (▼ icon only)
        ├── MenuItem (Salva e chiudi)
        └── MenuItem (Salva come template)
```

Icone comuni

Azioni generali:

- shaicons shaicons-save - Salva
- shaicons shaicons-x - Chiudi/Annulla
- shaicons shaicons-check - Conferma
- shaicons shaicons-plus - Aggiungi/Nuovo
- shaicons shaicons-edit - Modifica
- shaicons shaicons-trash-2 - Elimina
- shaicons shaicons-copy - Duplica
- shaicons shaicons-download - Scarica/Esporta
- shaicons shaicons-upload - Carica/Importa
- shaicons shaicons-refresh-cw - Aggiorna

Navigazione:

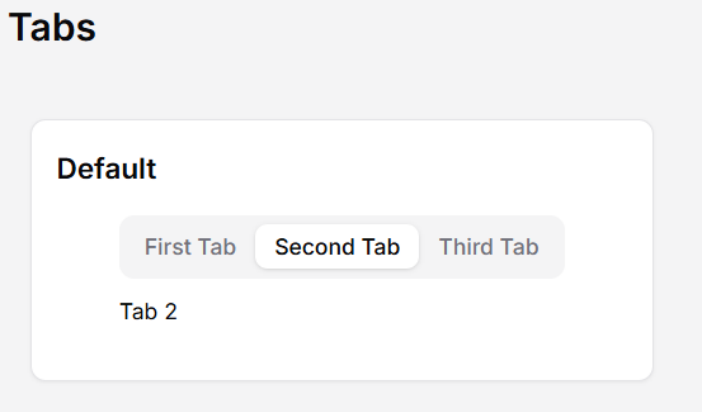
- shaicons shaicons-arrow-left - Indietro
- shaicons shaicons-arrow-right - Avanti
- shaicons shaicons-external-link - Apri esterno

Filtri e ricerca:

- shaicons shaicons-search - Cerca
- shaicons shaicons-filter - Filtro
- shaicons shaicons-sort-asc - Ordina crescente
- shaicons shaicons-sort-desc - Ordina decrescente

Le tabbed view

Le tabbed view permettono di organizzare contenuti correlati in schede separate, migliorando la navigazione all'interno di una singola pagina.



ShaTabs e ShaTab

ShaTabs è il contenitore delle schede, mentre *ShaTab* rappresenta una singola scheda.

ShaTabs

Proprietà principali:

Proprietà	Tipo	Descrizione
selectedTab	number	Indice della scheda selezionata (0-based)
className	string	Classi CSS personalizzate

Eventi:

- onChangeTab(event): Scatenato al cambio scheda
    - event.selectedTab: Nuovo indice selezionato

ShaTab

Proprietà principali:

Proprietà	Tipo	Descrizione
label	string	Etichetta della scheda
icon	string	Icona opzionale

Proprietà	Tipo	Descrizione
disabled	boolean	Disabilita la scheda
className	string	Classi CSS personalizzate

Struttura base

```
ShaTabs
├── ShaTab (Informazioni)
│     └── contenuto tab 1
├── ShaTab (Documenti)
│     └── contenuto tab 2
└── ShaTab (Storico)
      └── contenuto tab 3
```

Esempio implementazione:

```
ShaTabs
  selectedTab: 0
  ├── infoTab (ShaTab)
  │     label: "Informazioni"
  │     icon: "shaicons shaicons-info"
  │     └── ShaCard
  │           └── form informazioni
  │
  ├── docsTab (ShaTab)
  │     label: "Documenti"
  │     icon: "shaicons shaicons-file-text"
  │     └── ShaCard
  │           └── lista documenti
  │
  └── historyTab (ShaTab)
        label: "Storico"
        icon: "shaicons shaicons-clock"
        └── ShaCard
              └── timeline eventi
```

Gestione cambio tab

```
function* tabs_onChangeTab() {
  let selectedTab = this.view.tabs.selectedTab;

  // Carica dati specifici per la tab
  switch(selectedTab) {
    case 0: // Informazioni
      yield this.loadInfo();
      break;
    case 1: // Documenti
      yield this.loadDocuments();
      break;
    case 2: // Storico
      yield this.loadHistory();
```

```
      break;
  }
}
```

Pattern comuni

1. Form multi-step con tabs:

```
ShaTabs
  selectedTab: 0
  ├── ShaTab (Step 1: Dati base)
  │     └── form step 1 + pulsante "Avanti"
  │
  ├── ShaTab (Step 2: Dettagli)
  │     └── form step 2 + pulsanti "Indietro" e "Avanti"
  │
  └── ShaTab (Step 3: Conferma)
        └── riepilogo + pulsante "Salva"
```

```
// Avanza alla tab successiva
function* nextButton_onClick() {
  let currentTab = this.view.tabs.selectedTab;
  this.view.tabs.selectedTab = currentTab + 1;
}

// Torna alla tab precedente
function* prevButton_onClick() {
  let currentTab = this.view.tabs.selectedTab;
  this.view.tabs.selectedTab = currentTab - 1;
}
```

2. Lazy loading contenuti:

```
function* tabs_onChangeTab() {
  let tab = this.view.tabs.selectedTab;

  // Carica solo se non già caricato
  if (tab === 1 && !this.documentsLoaded) {
    this.view.docsSpinner.visible = true;
    yield this.view.documentsDM.load();
    this.view.docsSpinner.visible = false;
    this.documentsLoaded = true;
  }
}
```

3. Validazione prima di cambiare tab:

```
function* tabs_onChangeTab() {
  let newTab = this.view.tabs.selectedTab;
  let oldTab = this.currentTab || 0;

  // Se sta lasciando tab 0, valida
  if (oldTab === 0) {
    let isValid = yield this.validateStep1();

    if (!isValid) {
      // Torna alla tab precedente
      this.view.tabs.selectedTab = oldTab;
      yield app.toast("Completa tutti i campi", {
        variant: "destructive"
      });
      return;
    }
  }

  this.currentTab = newTab;
}
```

4. Badge con contatore su tab:

```
ShaTab
  label: `Documenti (${documentCount})`
  icon: "shaicons shaicons-file-text"
```

5. Tab disabilitate condizionalmente:

```
// Disabilita tab 2 e 3 finché non completi tab 1
function* step1ContinueButton_onClick() {
  yield this.saveStep1();

  // Abilita tab successiva
  this.view.step2Tab.disabled = false;
  this.view.tabs.selectedTab = 1;
}
```

6. Contenuto tab con DataMap:

```
// Tab con lista documenti
ShaTab
  label: "Documenti"
  └── ShaCard
        ├── ShaCardHeader
        │     └── ShaButton (Aggiungi documento)
        │
        └── ShaCardContent
              └── documentRow (template: documentsDM)
                    ├── documentName (ShaLabel)
                    ├── documentDate (ShaLabel)
                    └── actionsButton (ShaButton)
```

```
function* onLoad() {
  yield this.view.documentsDM.load();
}

function* documentRow_onClick() {
  let doc = this.sender.row;
  yield this.openDocument(doc.id);
}
```

7. Tabs con URL routing (opzionale):

```
// Apri view con tab specifica
yield view.push(App.ProjectDetailView, {
  title: "Progetto",
  projectId: id,
  initialTab: 1  // Apri su tab "Documenti"
});

// In ProjectDetailView onLoad
function* onLoad() {
  if (this.options.initialTab !== undefined) {
    this.view.tabs.selectedTab = this.options.initialTab;
  }
}
```

Stile tabs

Tabs orizzontali (default)

```
ShaTabs
  className: "w-full"
  ├── ShaTab (Tab 1)
  ├── ShaTab (Tab 2)
  └── ShaTab (Tab 3)
```

Tabs con icone

```
ShaTab
  label: "Documenti"
  icon: "shaicons shaicons-file-text"
```

Tabs compatte

```
ShaTabs
  className: "tabs-compact"
```

Best Practices

1. Numero di tabs: Mantieni 3-7 tabs per pagina (massimo leggibile)
2. Label chiare: Usa etichette brevi e descrittive

```
// ✅ Bene
"Informazioni", "Documenti", "Storico"

// ❌ Male
"Info generali del progetto", "Tutti i documenti", "Cronologia completa"
```

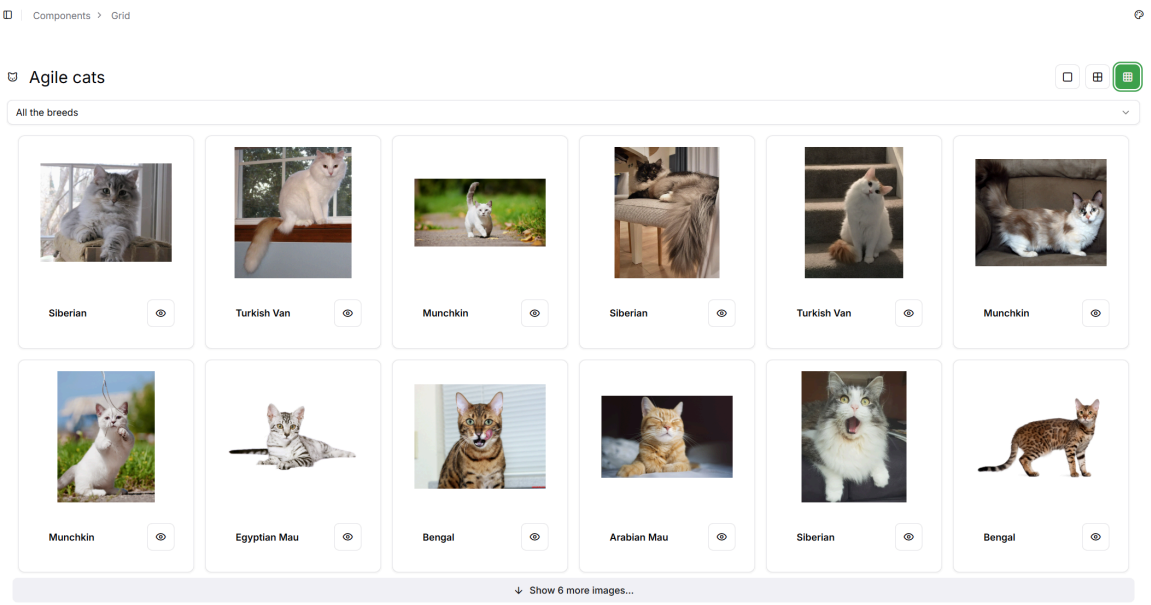
3. Caricamento lazy: Carica contenuti solo quando necessario
4. Stato tab: Mantieni traccia della tab corrente in una variabile
5. Validazione: Valida prima di permettere cambio tab in wizard
6. Feedback visivo: Mostra spinner durante caricamento contenuti tab

Definire il contenuto delle pagine

Il contenuto delle pagine in ShadowUI si organizza tramite un sistema di griglia responsive e card.

Sistema Grid

ShadowUI utilizza un sistema di griglia a *12 colonne* simile a *Bootstrap*, con breakpoint responsive.
Un esempio di griglia si può trovare nel progetto ShadowUI Design Patterns di cui vediamo un'immagine di seguito.



Componenti grid

ShaGrid: Contenitore principale

ShaRow: Riga che contiene colonne
ShaCol: Colonna responsive

ShaGrid

Contenitore con padding e margini gestiti.
Proprietà:

- style: Stili personalizzati
- className: Classi CSS

ShaRow

Contenitore orizzontale per colonne.
Proprietà:

Proprietà	Tipo	Descrizione
noWrap	boolean	Impedisce wrap delle colonne (default: false)
justifyContent	enum	Allineamento orizzontale: start, center, end, around, between
alignItems	enum	Allineamento verticale: start, center, end, stretch, baseline
className	string	Classi CSS personalizzate

ShaCol

Colonna responsive con breakpoint.
Proprietà breakpoint:

Breakpoint	Dimensione	Proprietà
xs	< 576px	xs
sm	≥ 576px	sm
md	≥ 768px	md
lg	≥ 992px	lg
xl	≥ 1200px	xl
xxl	≥ 1400px	xxl

Valori colonne:

- auto: Larghezza automatica
- fluid: Occupa spazio disponibile
- 1 a 12: Numero colonne (1/12 a 12/12)
- none: Nasconde la colonna

Esempio:

```
ShaRow
   ├── ShaCol
   │     xs: "12"       // 100% su mobile
   │     md: "6"        // 50% su tablet
   │     lg: "4"        // 33% su desktop
   │
   ├── ShaCol
   │     xs: "12"
   │     md: "6"
   │     lg: "4"
   │
   └── ShaCol
         xs: "12"
         md: "12"
         lg: "4"
```

Layout comuni

1. Layout a 2 colonne (50/50):

```
ShaGrid
   └── ShaRow
         ├── ShaCol (xs="12", md="6")
         │     └── contenuto sinistra
         │
         └── ShaCol (xs="12", md="6")
               └── contenuto destra
```

2. Layout sidebar + content (33/66):

```
ShaGrid
   └── ShaRow
         ├── ShaCol (xs="12", lg="4")
         │     └── sidebar/menu
         │
         └── ShaCol (xs="12", lg="8")
               └── contenuto principale
```

3. Layout 3 colonne uguali:

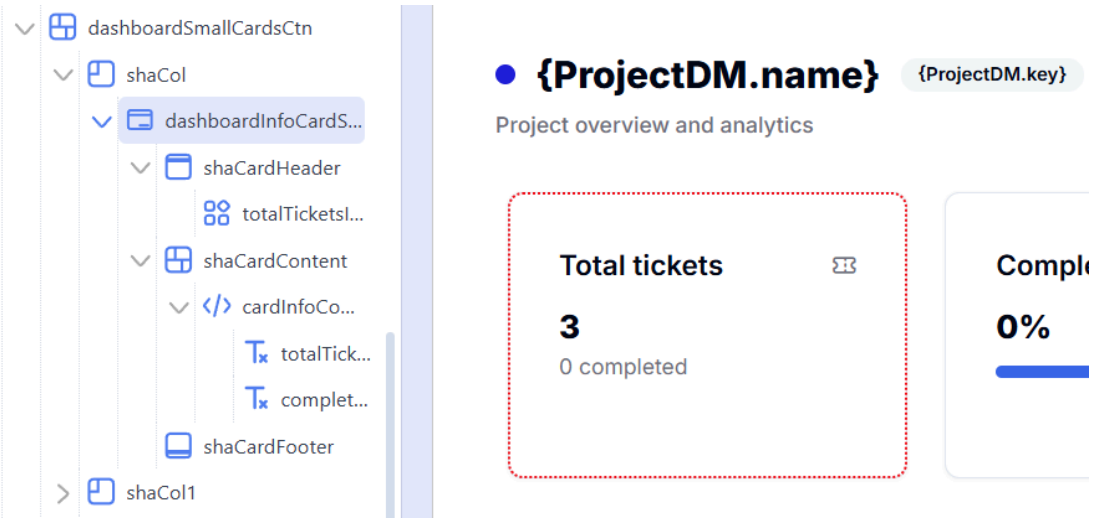
```
ShaGrid
   └── ShaRow
         ├── ShaCol (xs="12", md="6", lg="4")
         ├── ShaCol (xs="12", md="6", lg="4")
         └── ShaCol (xs="12", md="12", lg="4")
```

4. Layout dashboard con card:

```
ShaGrid
└── ShaRow
    ├── ShaCol (xs="12", sm="6", lg="3")
    │   └── ShaCard (Metric 1)
    │
    ├── ShaCol (xs="12", sm="6", lg="3")
    │   └── ShaCard (Metric 2)
    │
    ├── ShaCol (xs="12", sm="6", lg="3")
    │   └── ShaCard (Metric 3)
    │
    └── ShaCol (xs="12", sm="6", lg="3")
        └── ShaCard (Metric 4)
```

Card (ShaCard)

Le card sono contenitori per raggruppare contenuti correlati.



Struttura ShaCard

```
ShaCard
├── ShaCardHeader (opzionale)
├── ShaCardContent (obbligatorio)
└── ShaCardFooter (opzionale)
```

ShaCard

Contenitore principale.
Proprietà:

- className: Classi CSS (es. “shadow-lg”, “border-2”)
- style: Stili inline

ShaCardHeader

Intestazione della card.
Proprietà:

Proprietà	Tipo	Descrizione
title	string	Titolo principale
description	string	Sottotitolo/descrizione
className	string	Classi CSS personalizzate

Esempio:

```
ShaCardHeader
  title: "Dettagli Progetto"
  description: "Informazioni generali del progetto"
```

ShaCardContent

Contenuto principale della card.
Può contenere qualsiasi elemento: form, tabelle, liste, grafici, etc.

ShaCardFooter

Sezione footer con azioni o informazioni aggiuntive.
Uso tipico:

- Pulsanti che svolgono azioni (Salva, Annulla, Elimina)
- Timestamp (Ultima modifica, Creato il…)
- Link secondari

Pattern Card comuni

1. Card informazioni:

```
ShaCard
  ├── ShaCardHeader
  │     title: "Informazioni Utente"
  │
  ├── ShaCardContent
  │     ├── ShaLabel (Nome)
  │     ├── ShaLabel (Email)
  │     └── ShaLabel (Ruolo)
  │
  └── ShaCardFooter
        └── ShaButton (Modifica)
```

2. Card form:

```
ShaCard
   ├── ShaCardHeader
   │     title: "Nuovo Progetto"
   │     description: "Compila i campi per creare un nuovo progetto"
   │
   ├── ShaCardContent
   │     ├── ShaInput (Nome)
   │     ├── ShaInput (Descrizione)
   │     └── ShaSelect (Categoria)
   │
   └── ShaCardFooter
         ├── ShaButton (Salva - default)
         └── ShaButton (Annulla - outline)
```

3. Card dashboard con metrica:

```
ShaCard
   ├── ShaCardHeader
   │     title: "Vendite Totali"
   │
   └── ShaCardContent
         ├── ShaLabel (€ 145,000 - size: "xxl")
         └── ShaLabel (+12% rispetto al mese scorso - muted)
```

4. Card lista dinamica con DataMap:

```
ShaCard
   ├── ShaCardHeader
   │     title: "Progetti Attivi"
   │     action: ShaButton (+ Nuovo)
   │
   ├── ShaCardContent
   │     └── projectRow (template: projectsDM)
   │           ├── ShaLabel (§name)
   │           ├── ShaBadge (§status)
   │           └── ShaButton (Apri - ghost)
   │
   └── ShaCardFooter
         └── ShaLabel (Totale: §projectsDM.rows.length progetti)
```

5. Card nidificate:

```
ShaCard (contenitore principale)
   └── ShaCardContent
         ├── ShaCard (sezione 1)
         │     └── contenuto 1
         │
         └── ShaCard (sezione 2)
               └── contenuto 2
```

6. Card con grafico:

```
ShaCard
   ├── ShaCardHeader
   │     title: "Andamento Vendite"
   │
   ├── ShaCardContent
   │     └── ShaChart
   │           type: "line"
   │           data: salesData
   │
   └── ShaCardFooter
         └── ShaButton (Esporta dati)
```

Layout completo esempio

Pagina dettaglio progetto:

```
ShaPage
   └── ShaGrid
         ├── ShaRow (Header con azioni)
         │     └── ShaCol
         │           ├── ShaLabel (Titolo progetto - xxl)
         │           └── ShaRow (Pulsanti)
         │
         ├── ShaRow (Contenuto principale)
         │     ├── ShaCol (xs="12", lg="8")
         │     │     ├── ShaCard (Informazioni)
         │     │     ├── ShaCard (Descrizione)
         │     │     └── ShaCard (Team)
         │     │
         │     └── ShaCol (xs="12", lg="4")
         │           ├── ShaCard (Stato)
         │           ├── ShaCard (Date)
         │           └── ShaCard (Tag)
         │
         └── ShaRow (Footer)
               └── ShaCol
                     └── ShaCard (Storico attività)
```

Best Practices

1. **Mobile-first**: Inizia sempre con xs="12" per mobile
2. **Breakpoint progressivi**: Definisci layout per ogni dimensione
   ```
   xs="12"  // Mobile: 100%
   md="6"   // Tablet: 50%
   lg="4"   // Desktop: 33%
   ```
3. **Card coerenti**: Mantieni stile uniforme tra card
4. **Spazi consistenti**: Usa classi gap per spacing
   ```
   ShaRow
     className: "gap-4"  // 1rem di spazio tra colonne
   ```
5. **Padding card**: Usa ShaCardContent per padding automatico

6. **Card max-width**: Limita larghezza card per leggibilità
   ```
   ShaCard
     className: "max-w-2xl mx-auto"
   ```

Il page controller

Il page controller gestisce la navigazione e il ciclo di vita delle pagine nell'applicazione.

Metodi principali

I metodi principali del page controller sono già stati descritti nel capitolo Il contenitore pagine (ShaNavController) nella sezione "Navigazione: Proprietà e Metodi del Page Controller ". Qui riassumiamo i concetti chiave e aggiungiamo pattern avanzati.

Riepilogo metodi

Metodo	Descrizione
push(viewclass, options)	Apri una nuova pagina
pop(count, info)	Torna indietro
remove(count, info)	Rimuovi pagine dallo stack
toggleMenu(show)	Apri/chiudi sidebar
setTitle(title)	Aggiorna titolo (solo Sticky)
setBreadcrumb()	Aggiorna breadcrumb
getActivePage()	Ottieni pagina attiva
postMessage(msg)	Invia messaggio alle pagine

Pattern avanzati

1. Navigazione condizionale

```
function* openDetail() {
  // Controlla permessi prima di navigare
  let hasPermission = yield app.checkPermission("view_details");

  if (!hasPermission) {
    yield app.toast("Accesso negato", { variant: "destructive" });
    return;
  }

  yield view.push(App.DetailView, {
    title: "Dettaglio",
    itemId: this.itemId
  });
}
```

2. Navigazione con parametri complessi

```
function* openEditView() {
  yield view.push(App.EditView, {
    title: "Modifica Ticket",
    // Passa oggetto completo
    ticket: {
      id: this.ticket.id,
      title: this.ticket.title,
      status: this.ticket.status,
      assignee: this.ticket.assignee
    },
    // Callback per aggiornamento
    onSave: (updatedTicket) => {
      this.updateLocalTicket(updatedTicket);
    }
  });
}

// In EditView - onLoad
function* onLoad() {
  let ticket = this.options.ticket;

  // Popola form
  this.view.titleInput.value = ticket.title;
  this.view.statusSelect.value = ticket.status;
}

// In EditView - save
function* saveButton_onClick() {
  let updated = yield this.save();

  // Chiama callback se fornito
  if (this.options.onSave) {
    this.options.onSave(updated);
  }

  yield view.pop(1, { saved: true });
}
```

3. Stack profondo con breadcrumb

```
// Crea navigazione profonda mantenendo breadcrumb

// 1. Dashboard (root)
yield view.push(App.DashboardView, {
  root: true,
  title: "Dashboard"
});
```

```
// 2. Lista progetti
yield view.push(App.ProjectsView, {
  title: "Progetti"
});


// Breadcrumb: Dashboard > Progetti

// 3. Dettaglio progetto
yield view.push(App.ProjectDetailView, {
  title: "Progetto Alpha",
  projectId: 123
});
// Breadcrumb: Dashboard > Progetti > Progetto Alpha

// 4. Lista task
yield view.push(App.TasksView, {
  title: "Task",
  projectId: 123
});
// Breadcrumb: Dashboard > Progetti > Progetto Alpha > Task

// 5. Dettaglio task
yield view.push(App.TaskDetailView, {
  title: "Task #45",
  taskId: 45
});
// Breadcrumb: Dashboard > ... > Progetto Alpha > Task > Task #45
```

4. Rimozione selettiva dallo stack

```
// Scenario: Dashboard → Projects → ProjectDetail → TaskList →
TaskDetail
// Vogliamo: Dashboard → ProjectDetail → TaskDetail
// (rimuoviamo Projects e TaskList)

function* saveTaskButton_onClick() {
  yield this.saveTask();

  // Rimuovi TaskList e Projects dallo stack
  yield view.remove(App.TasksView);
  yield view.remove(App.ProjectsView);

  // Ora lo stack è: Dashboard → ProjectDetail → TaskDetail

  yield app.toast("Task salvato", { variant: "success" });
}
```

5. Navigazione modale con popup

```
// Apri form come popup invece che pagina normale
function* quickAddButton_onClick() {
  yield view.push(App.QuickAddView, {
    popup: true,
    title: "Aggiungi rapido",
    // Opzioni popup
    popupPosition: "center",
    popupWidth: "500px",
    backdrop: true
  });
}

// In QuickAddView - dopo salvataggio
function* saveButton_onClick() {
  yield this.save();

  // Chiudi popup
  this.view.close();

  // Notifica pagina sottostante
  yield view.postMessage({
    type: "item-added",
    item: this.savedItem
  });
}

// In pagina principale - ricevi notifica
function* onMessage(message) {
  if (message.type === "item-added") {
    // Ricarica lista
    yield this.view.itemsDM.load();
  }
}
```

6. Gestione stati back button

```
// Gestisci back button browser/Android
function* canClose() {
  // Chiamato quando utente preme back
  let hasUnsavedChanges = this.checkUnsavedChanges();

  if (hasUnsavedChanges) {
    let confirm = yield app.popup(
      "Modifiche non salvate",
      "Vuoi salvare prima di uscire?",
      ["Annulla", "Esci senza salvare", "Salva ed esci"]
    );

    if (confirm === 0) {
      return false; // Blocca chiusura
    } else if (confirm === 2) {
      yield this.save();
    }
  }
  return true; // Permetti chiusura
}
```

7. Preload pagine successive

```
// Precarica dati della pagina successiva per transizioni veloci
function* openDetailButton_onClick() {
  let itemId = this.selectedItem.id;

  // Avvia caricamento dati in background
  let dataPromise = app.api.getItemDetail(itemId);

  // Naviga immediatamente
  yield view.push(App.DetailView, {
    title: "Dettaglio",
    itemId: itemId,
    preloadedData: dataPromise  // Passa promise
  });
}

// In DetailView - onLoad
function* onLoad() {
  if (this.options.preloadedData) {
    // Usa dati precaricati
    let data = yield this.options.preloadedData;
    this.populateView(data);
  } else {
    // Carica normalmente
    yield this.loadData();
  }
}
```

8. Navigazione con transizioni personalizzate

```
// Disabilita animazione per navigazione veloce
yield view.push(App.TargetView, {
  title: "Target",
  animate: false  // Nessuna transizione
});

// Forza animazione anche se disabilitata globalmente
yield view.push(App.TargetView, {
  title: "Target",
  animate: true
});
```

9. Comunicazione bidirezionale tra pagine

```
// Pagina A apre Pagina B e attende risultato

// In PageA
function* selectUserButton_onClick() {
  yield view.push(App.UserPickerView, {
    popup: true,
    title: "Seleziona Utente",
    currentUser: this.currentUser
  });
}

function* onMessage(message) {
  if (message.type === "user-selected") {
    // Ricevi utente selezionato
    this.view.userInput.value = message.user.name;
    this.selectedUser = message.user;
  }
}

// In UserPickerView
function* userRow_onClick() {
  let user = this.sender.row;

  // Invia risultato a pagina chiamante
  yield view.postMessage({
    type: "user-selected",
    user: user
  });

  // Chiudi picker
  this.view.close();
}
```

10. Gestione errori navigazione

```
function* navigateToDetail() {
  try {
    let itemId = this.selectedItem.id;

    // Verifica esistenza item
    let exists = yield app.api.checkItemExists(itemId);

    if (!exists) {
      yield app.toast("Elemento non trovato", {
        variant: "destructive"
      });
      return;
    }

    // Naviga
    yield view.push(App.DetailView, {
      title: "Dettaglio",
      itemId: itemId
    });

  } catch (error) {
    yield app.toast(`Errore: ${error.message}`, {
      variant: "destructive"
    });

    console.error("Navigation error:", error);
  }
}
```

Ciclo di vita pagina

Eventi principali:

1. **Constructor**: Creazione istanza
2. **onLoad(options)**: Caricamento iniziale
3. **onShow()**: Pagina mostrata (anche dopo ritorno)
4. **onBack(closedView, info)**: Ricezione da pagina chiusa
5. **onMessage(message)**: Ricezione messaggi
6. **canClose()**: Verifica prima di chiusura
7. **onClose()**: Chiusura pagina

Esempio completo:

```
// In una view
function* onLoad() {
  console.log("1. onLoad - caricamento iniziale");

  // Ricevi parametri
  this.itemId = this.options.itemId;

  // Carica dati
  yield this.loadData();
}

function* onShow() {
  console.log("2. onShow - pagina mostrata");

  // Aggiorna dati se necessario
  if (this.needsRefresh) {
    yield this.refresh();
    this.needsRefresh = false;
  }
}

function* onBack(closedView, info) {
  console.log("3. onBack - ritorno da altra pagina");

  // Ricevi dati da pagina chiusa
  if (info.saved) {
    yield this.refresh();
  }
}
```

```
function* onMessage(message) {
  console.log("4. onMessage - messaggio ricevuto");

  if (message.type === "data-changed") {
    this.needsRefresh = true;
  }
}

function* canClose() {
  console.log("5. canClose - verifica chiusura");

  // Verifica modifiche non salvate
  if (this.hasUnsavedChanges()) {
    let confirm = yield app.popup(
      "Conferma",
      "Ci sono modifiche non salvate. Continuare?",
      ["Annulla", "Esci"]
    );

    return (confirm === 1);
  }

  return true;
}

function* onClose() {
  console.log("6. onClose - chiusura pagina");

  // Cleanup risorse
  this.cleanup();
}
```

Sidebar e Menu

La sidebar contiene il menu di navigazione principale dell'applicazione.

Componenti Sidebar

ShaSidebar

Contenitore principale della sidebar.

Proprietà:

Proprietà	Tipo	Descrizione
variant	enum	Stile: classic, floating, inset
side	enum	Posizione: left, right
visible	boolean	Mostra/nascondi sidebar
iconCollapseMode	boolean	Modalità icone quando collassata

ShaSidebarHeader

Sezione header della sidebar (logo, titolo).

ShaSidebarContent

Sezione centrale con i menu di navigazione.

ShaSidebarFooter

Sezione footer (impostazioni, profilo).

ShaSidebarGroup

Raggruppa voci di menu correlate.
Proprietà:

Proprietà	Tipo	Descrizione
label	string	Etichetta del gruppo
variant	enum	Tipo: group, collapsible, subgroup, subcollapsible
state	enum	Stato collassabile: open, closed
action	string	Icona pulsante azione (es. per aggiungere)
icon	string	Icona del gruppo

Eventi:

- onClick(event): Click sul gruppo collassabile
- onAction(event): Click sul pulsante azione
- onToggleState(event): Cambio stato aperto/chiuso

Varianti gruppo

1. Group (standard)

```
ShaSidebarGroup
  label: "Navigazione"
  variant: "group"
  ├── ShaSidebarItem (Dashboard)
  ├── ShaSidebarItem (Progetti)
  └── ShaSidebarItem (Impostazioni)
```

2. Collapsible (espandibile/collassabile)

```
ShaSidebarGroup
  label: "Admin"
  variant: "collapsible"
  state: "closed"
  ├── ShaSidebarItem (Utenti)
  ├── ShaSidebarItem (Ruoli)
  └── ShaSidebarItem (Log)
```

3. Subgroup (sottogruppo senza border)

```
ShaSidebarGroup
  label: "Sottosezione"
  variant: "subgroup"
  ├── ShaSidebarItem (...)
  └── ShaSidebarItem (...)
```

4. Subcollapsible (sottogruppo collassabile)

```
ShaSidebarGroup
  label: "Avanzate"
  variant: "subcollapsible"
  state: "closed"
  ├── ShaSidebarItem (...)
  └── ShaSidebarItem (...)
```

ShaSidebarItem

Singola voce di menu cliccabile.
Proprietà:

Proprietà	Tipo	Descrizione
label	string	Testo della voce
icon	string	Icona (es. “shaicons shaicons-home”)
iconPosition	enum	Posizione: left, right
disabled	boolean	Disabilita la voce
className	string	Classi CSS personalizzate

Eventi:

- onClick(event): Click sulla voce

Menu statico

```
ShaSidebarContent
   ├── ShaSidebarGroup
   │     label: "Navigazione"
   │     variant: "group"
   │     ├── dashboardItem (ShaSidebarItem)
   │     │     label: "Dashboard"
   │     │     icon: "shaicons shaicons-layout-dashboard"
   │     │
   │     ├── projectsItem (ShaSidebarItem)
   │     │     label: "Progetti"
   │     │     icon: "shaicons shaicons-folder"
   │     │
   │     └── tasksItem (ShaSidebarItem)
   │           label: "Task"
   │           icon: "shaicons shaicons-check-square"
   │
   └── ShaSidebarGroup
         label: "Admin"
         variant: "collapsible"
         state: "closed"
         ├── usersItem (ShaSidebarItem)
         │     label: "Utenti"
         │     icon: "shaicons shaicons-users"
         │
         └── settingsItem (ShaSidebarItem)
               label: "Impostazioni"
               icon: "shaicons shaicons-settings"
```

Handler navigazione:

```
function* dashboardItem_onClick() {
  yield view.push(App.DashboardView, {
    root: true,
    title: "Dashboard"
  });

  if (app.device.isMobile) {
    yield view.toggleMenu(false);
  }
}

function* projectsItem_onClick() {
  yield view.push(App.ProjectsView, {
    root: true,
    title: "Progetti"
  });

  if (app.device.isMobile) {
    yield view.toggleMenu(false);
  }
}
```

Menu dinamico con DataMap

```
ShaSidebarGroup
  label: "Sprint"
  variant: "collapsible"
  state: "open"
  action: "shaicons shaicons-plus"  // Pulsante "Nuovo sprint"
  └── sprintItem (ShaSidebarItem)
        template: sprintDM
        label: "§name"
        icon: "shaicons shaicons-git-branch"
```

```
// MainPage - onLoad
function* onLoad() {
  yield this.view.sprintDM.load();
}

// sprintDM - onRowComposition
function* sprintDM_onRowComposition(row) {
  // Aggiungi badge con conteggio task
  let item = this.sender;
  item.badge = row.taskCount;
}

// sprintItem - onClick
function* sprintItem_onClick() {
  let sprint = this.sender.row;

  yield view.push(App.SprintDetailView, {
    title: sprint.name,
    sprintId: sprint.id
  });

  if (app.device.isMobile) {
    yield view.toggleMenu(false);
  }
}

// sprintGroup - onAction (click su pulsante +)
function* sprintGroup_onAction() {
  yield view.push(App.NewSprintView, {
    popup: true,
    title: "Nuovo Sprint"
  });
}
```

Pattern comuni

1. Evidenziazione voce attiva

```
function* dashboardItem_onClick() {
  // Rimuovi classe attiva da tutte le voci
  this.view.dashboardItem.className = "";
  this.view.projectsItem.className = "";
  this.view.tasksItem.className = "";

  // Aggiungi classe attiva alla voce corrente
  this.view.dashboardItem.className = "bg-accent";

  yield view.push(App.DashboardView, {
    root: true,
    title: "Dashboard"
  });
}
```

2. Badge con contatori

```
ShaSidebarItem
  label: "Notifiche"
  icon: "shaicons shaicons-bell"
  badge: "5"  // Badge con numero

function* onLoad() {
  // Aggiorna contatore notifiche
  let count = yield app.api.getUnreadNotifications();
  this.view.notificationsItem.badge = count > 0 ? count.toString() : null;
}
```

3. Menu con permessi

```
function* onLoad() {
  // Mostra voci solo se utente ha permessi
  let isAdmin = yield app.user.hasRole("admin");

  this.view.adminGroup.visible = isAdmin;
  this.view.usersItem.visible = isAdmin;
  this.view.settingsItem.visible = isAdmin;
}
```

4. Sottomenu nidificati

```
ShaSidebarGroup (Projects)
  variant: "collapsible"
  ├── ShaSidebarItem (All Projects)
  │
  └── ShaSidebarGroup (Active)
       variant: "subcollapsible"
       ├── ShaSidebarItem (Project A)
       ├── ShaSidebarItem (Project B)
       └── ShaSidebarItem (Project C)
```

5. Separatori tra gruppi

```
ShaSidebarContent
   ├── ShaSidebarGroup (Navigation)
   ├── ShaSeparator
   ├── ShaSidebarGroup (Projects)
   ├── ShaSeparator
   └── ShaSidebarGroup (Admin)
```

6. Footer con profilo utente

```
ShaSidebarFooter
   └── ShaRow
         ├── ShaAvatar
         │     src: "§app.user.avatar"
         │     fallback: "§app.user.initials"
         │
         └── ShaCol
               ├── ShaLabel (§app.user.name)
               └── ShaLabel (§app.user.email - muted)
```

Icone menu comuni

Navigazione:

- shaicons shaicons-layout-dashboard - Dashboard
- shaicons shaicons-folder - Progetti/Cartelle
- shaicons shaicons-file-text - Documenti
- shaicons shaicons-check-square - Task/Todo
- shaicons shaicons-calendar - Calendario/Eventi

Gestione:

- shaicons shaicons-users - Utenti/Team
- shaicons shaicons-settings - Impostazioni
- shaicons shaicons-database - Dati/Database
- shaicons shaicons-package - Moduli/Pacchetti

Azioni:

- shaicons shaicons-plus - Aggiungi/Nuovo
- shaicons shaicons-search - Cerca
- shaicons shaicons-filter - Filtri
- shaicons shaicons-download - Download/Esporta
- shaicons shaicons-upload - Upload/Importa

Tempo:

- shaicons shaicons-clock - Storico/Timeline
- shaicons shaicons-git-branch - Sprint/Versioni
- shaicons shaicons-trending-up - Statistiche/Report

Documenti:

- shaicons shaicons-file - File generico

- shaicons shaicons-file-text - Documento testo
- shaicons shaicons-image - Immagini
- shaicons shaicons-archive - Archivio

Componenti Shadow UI

ShadowUI offre oltre 60 componenti pronti all'uso, organizzati per categoria.

1. Controlli Form

ShaInput

Campo di input testuale.
Proprietà:

- value - Valore (data-bindable)
- label - Etichetta
- labelPosition - top, left, none
- placeholder - Testo placeholder
- hintText - Testo di aiuto
- disabled - Disabilitato

Eventi:

- onChange(event)

ShaCheckbox

Casella di selezione.
Proprietà:

- checked - Stato (data-bindable)
- label - Etichetta
- labelSide - left, right - disabled

Eventi:

- onChange(event)

ShaSwitch

Interruttore on/off.
Proprietà:

- checked - Stato (data-bindable)
- label - Etichetta
- labelSide - left, right - disabled

Eventi:

- onChange(event)

ShaSelect

Menu a tendina.
Proprietà:

- value - Valore selezionato (data-bindable)
- placeholder - Testo placeholder
- label - Etichetta interna
- list - Enum con opzioni - disabled

Eventi:

- onSelect(event)

ShaRadioGroup

Gruppo di radio button per selezione esclusiva.
Proprietà:

- value - Valore selezionato (data-bindable)
- layout - vertical, horizontal

ShaRadioItem:

- value - Valore opzione
- label - Etichetta - disabled

Eventi:

- onValueChange(event)

ShaCombobox

Select con ricerca integrata.
Proprietà:

- value - Valore selezionato (data-bindable)
- placeholder - list
- Enum opzioni - searchEnabled
- Abilita ricerca - disabled

Eventi:

- onSelect(event)
- onSearch(event)

2. Widget Interattivi

ShaDatePicker

Selettore date.
Proprietà:

- selectedDate1 - Data selezionata (o inizio range)
- selectedDate2 - Data fine range (se mode=“range”)
- mode - single, range

- locale - Locale (es. “it-IT”)
- placeholder - dateFormat
- Formato data - resetButtonVisible
- Mostra reset - disabled

Metodi:
- setSelected(date)
- getSelected(callback)

Eventi:
- onSelect(event)

ShaSlider

Cursore per valori numerici.
Proprietà:
- value - Valore(i) (data-bindable)
- min - Minimo
- max - Massimo
- step - Incremento
- knobsNumber - 1 o 2 cursori
- disabled

Eventi:
- onInput(event) - Durante movimento
- onChange(event) - Al rilascio

ShaProgress

Barra di progresso.
Proprietà:
- value - Valore corrente (0-100)
- max - Valore massimo
- showLabel - Mostra percentuale

ShaInputOTP

Input per codici OTP.
Proprietà:
- value - Codice inserito
- length - Numero cifre (default: 6)
- separator - minus, dot, none
- mode - numberOnly, charOnly, charAndNumber
- disabled

Eventi:
- onChange(event)
- onComplete(event) - Tutti i campi compilati

ShaToggle

Pulsante toggle con stato on/off.
Proprietà:

- pressed - Stato (data-bindable)
- text - Testo
- icon - Icona
- iconPosition - left, right, only
- variant - Stile
- disabled

Eventi:

- onClick(event)

3. Visualizzazione Dati

ShaChart

Grafici versatili (6 tipi).
Proprietà:

- type - line, bar, area, pie, radar, radial
- data - Array dati
- dataKeys - Chiavi da plottare (line/bar/area/radar)
- labels - Etichette legenda
- xAxisKey - Proprietà asse X (default: “name”)
- nameKey - Proprietà nome (pie/radial)
- dataKey - Proprietà valore (pie/radial)
- showTooltip - Mostra tooltip (default: true)
- showGrid - Mostra griglia (default: false)
- showLegend - Mostra legenda (default: true)

ShaCalendar

Calendario per selezione date ed eventi.
Proprietà:

- selectedDate - Data selezionata
- mode - single, multiple, range
- locale - Locale
- events - Array eventi
- disabled
- Date disabilitate

Eventi:

- onDateSelect(event)

ShaTableElement

Tabelle HTML strutturate.

Varianti:

- table - Contenitore
- tableHead - Header
- tableBody - Body
- tableFoot - Footer
- tableRow - Riga
- tableHeader - Cella header (th)
- tableData - Cella dati (td)
- caption - Didascalia

4. Contenuti e Badge

ShaBadge

Badge compatto.
Proprietà:

- label - Testo (data-bindable)
- variant - default, secondary, outline, destructive

ShaAvatar

Avatar utente.
Proprietà:

- src - URL immagine
- lt - Testo alternativo
- fallback - Iniziali se no immagine
- size - sm, md, lg

ShaLabel

Etichetta testuale.
Proprietà:

- innerText - Testo
- size - default, small, medium, large, xl, xxl
- aspect - default, muted, primary, secondary, accent, destructive

ShaSkeleton

Placeholder caricamento.
Proprietà:

- width - Larghezza
- height - Altezza

ShaSeparator

Linea separatrice.
Proprietà:

- orientation - horizontal, vertical

ShaAlertBadge

Badge di alert.
Proprietà:

- label - Testo
- variant - default, destructive
- icon - Icona

5. Layout Avanzati

ShaAccordion

Pannelli espandibili.
Proprietà:

- title - Titolo
- state - open, closed

Eventi:

- onToggle(event)

ShaCarousel

Carosello contenuti.
Proprietà:

- direction - horizontal, vertical
- autoPlay - Avanza automaticamente
- interval - Intervallo auto-play (ms)
- showDots - Mostra indicatori
- showArrows - Mostra frecce

Eventi:

- nChange(event) - Cambio slide

Riepilogo componenti per caso d’uso

Form di input: ShaInput, ShaCheckbox, ShaSwitch, ShaSelect, ShaRadioGroup, ShaCombobox, ShaDatePicker.

Azioni: ShaButton, ShaToggle, ShaDropdownMenu.

Navigazione: ShaTabs, ShaBreadcrumb, ShaSidebar, ShaNavController.

Layout: - ShaGrid, ShaRow, ShaCol, ShaCard, ShaAccordion.

Dati: ShaChart, ShaCalendar, ShaTableElement, ShaProgress.

Contenuti: ShaLabel, ShaBadge, ShaAvatar, ShaIcon, ShaSeparator, ShaSkeleton

Per esempi dettagliati di ogni componente, consultare i progetti shadow-design-patterns e Sprinty nella Console di Instant Developer Cloud.

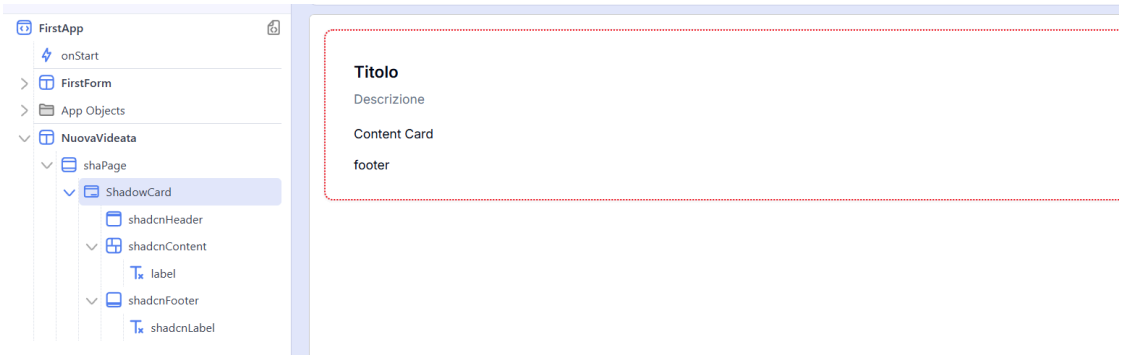
Template

Nella libreria Shadow è presente la videata *Template* che contiene i seguenti elementi di design:

- ShadowCard
- ShadowTable
- ShadowTabs
- ShadowRadioGroup
- ShadowDropdownMenu
- ShadowMenuBar

ShadowCard

Implementa un esempio di card con tutti gli elementi che solitamente si di solito si utilizzano come intestazione contenuto e footer



Personalizzazione

ShadowUI è altamente personalizzabile tramite temi, CSS Variables e Tailwind CSS.

Temi

ShadowUI supporta due temi predefiniti: **Light** e **Dark**.
Indipendentemente da essi è possibile scegliere un tema colorato da applicare all'applicazione.

Cambio mode tema

```
// Ottieni tema corrente
let currentTheme = app.theme.darkMode; // dark true o false

// Toggle tema
let newTheme = currentTheme === true ? false : true;
app.updateTheme();
```

```
// Imposta tema dark
app.theme.darkMode = true;

app.updateTheme();
```

Cambio tema colore

```
// Ottieni tema corrente
let currentTheme = app.theme.mainColor;
// stringa contenente il colore. Default “blue”


// Imposta tema verde
Let newTheme = {mainColor : App.themes.green};

// Costante contenente il valore stringa “green”
app.updateTheme(newTheme);
```

CSS Variables

ShadowUI usa CSS Variables per colori e spacing. È possibile personalizzarle nel file CSS del progetto.

Colori principali

```
:root {
  /* Background */
  --background: 0 0% 100%;
  --foreground: 222.2 84% 4.9%;

  /* Primary */
  --primary: 221.2 83.2% 53.3%;
  --primary-foreground: 210 40% 98%;

  /* Secondary */
  --secondary: 210 40% 96.1%;
  --secondary-foreground: 222.2 47.4% 11.2%;

  /* Accent */
  --accent: 210 40% 96.1%;
  --accent-foreground: 222.2 47.4% 11.2%;

  /* Destructive */
  --destructive: 0 84.2% 60.2%;
  --destructive-foreground: 210 40% 98%;

  /* Muted */
  --muted: 210 40% 96.1%;
  --muted-foreground: 215.4 16.3% 46.9%;

  /* Border */
  --border: 214.3 31.8% 91.4%;
  --input: 214.3 31.8% 91.4%;
```

```
  /* Ring (focus) */
  --ring: 221.2 83.2% 53.3%;

  /* Radius */
  --radius: 0.5rem;
}

.dark {
  --background: 222.2 84% 4.9%;
  --foreground: 210 40% 98%;

  --primary: 217.2 91.2% 59.8%;
  --primary-foreground: 222.2 47.4% 11.2%;

  /* ... altri colori per dark mode */
}
```

Personalizzare i colori

Crea un file custom-theme.css nel tuo progetto:

```
/* Custom color scheme */
:root {
  /* Brand colors */
  --primary: 142 76% 36%;  /* Verde custom */
  --primary-foreground: 0 0% 100%;

  /* Custom accent */
  --accent: 43 96% 56%;  /* Giallo custom */
  --accent-foreground: 0 0% 0%;
}

/* Custom button style */
.btn-custom {
  background-color: hsl(var(--primary));
  color: hsl(var(--primary-foreground));
  border-radius: calc(var(--radius) * 2);
}

/* Custom card style */
.card-elevated {
  box-shadow: 0 10px 25px -5px rgba(0, 0, 0, 0.1);
  border: none;
}
```

Importa il file CSS nel progetto tramite l’IDE.

Tailwind CSS

ShadowUI include Tailwind CSS. Puoi usare tutte le utility classes.

Utility comuni

Spacing:

```
className: "p-4 m-2 gap-4"
// padding: 1rem
// margin: 0.5rem
// gap: 1rem
```

Layout:

```
className: "flex items-center justify-between"
className: "grid grid-cols-3 gap-4"
```

Colori:

```
className: "bg-primary text-primary-foreground"
className: "border-destructive text-destructive"
```

Typography:

```
className: "text-2xl font-bold text-muted-foreground"
```

Responsive:

```
className: "w-full md:w-1/2 lg:w-1/3"
```

Classi personalizzate

Per componenti specifici

```
// Button con style custom
ShaButton
  text: "Custom Button"
  className: "btn-custom hover:scale-105 transition-transform"

// Card con ombra elevata
ShaCard
  className: "card-elevated backdrop-blur-sm"

// Input con bordo colorato
ShaInput
  className: "border-2 border-primary focus:border-accent"
```

Icon set personalizzati

ShadowUI usa *Lucide Icons* di default. È possibile aggiungere altri icon set.

Aggiungere Font Awesome

1. Importa Font Awesome CSS nel progetto
2. Usa le classi nei componenti:

```
ShaButton
  icon: "fas fa-save"  // Font Awesome
  text: "Salva"

ShaSidebarItem
  icon: "fas fa-home"
  label: "Home"
```

Aggiungere Material Icons

1. Importare Material Icons CSS
2. Usa le classi:

```
ShaButton
  icon: "material-icons"
  text: "save"  // Nome icona Material

// Oppure usa uno span custom
<span class="material-icons">home</span>
```

Componenti custom

È possibile creare componenti riutilizzabili estendendo quelli esistenti.

Esempio: Button con icona e badge

```
// Crea classe CustomButton estendendo ShaButton
CustomButton extends ShaButton
   ├── icon (ShaIcon)
   ├── text (ShaLabel)
   └── badge (ShaBadge)

// Usa CustomButton
CustomButton
  text: "Notifiche"
  icon: "shaicons shaicons-bell"
  badgeCount: 5
```

Esempio: Card statistica

```
// Componente riutilizzabile StatCard
ShaCard (className: "stat-card")
   ├── ShaCardHeader
   │      ├── ShaLabel (Titolo - muted)
   │      └── ShaIcon (Icona)
   │
   └── ShaCardContent
          ├── ShaLabel (Valore - xxl)
          └── ShaLabel (Variazione - small)
```

```
// CSS custom per stat-card
.stat-card {
  border-left: 4px solid hsl(var(--primary));
  transition: transform 0.2s;
}
.stat-card:hover {
  transform: translateY(-2px);
  box-shadow: 0 4px 12px rgba(0, 0, 0, 0.1);
}
```

Animazioni custom

Aggiungi animazioni con Tailwind:

```
// Button con animazione
ShaButton
  className: "hover:scale-105 active:scale-95 transition-all duration-200"

// Card con fade-in
ShaCard
  className: "animate-fade-in"

// Skeleton pulse
ShaSkeleton
  className: "animate-pulse"
```

Definisci animazioni custom in CSS:

```
@keyframes fade-in {
  from {
    opacity: 0;
    transform: translateY(10px);
  }
  to {
    opacity: 1;
    transform: translateY(0);
  }
}

.animate-fade-in {
  animation: fade-in 0.3s ease-out;
}
```

Best Practices personalizzazione

1. **Usa CSS Variables**: Per colori e spacing riutilizzabili
2. **Mantieni coerenza**: Usa lo stesso design system in tutta l'app
3. **Responsive**: Testa sempre su mobile, tablet, desktop
4. **Accessibilità**: Mantieni contrasti sufficienti per i colori
5. **Performance**: Limita animazioni pesanti
6. **Dark mode**: Testa sempre entrambi i temi