readme.md for @serve.zone/catalog

The complete UI component library for **serve.zone** — a full-featured admin and management interface for onebox server management, built as a collection of reusable web components.

Install

```
pnpm install @serve.zone/catalog
```

Issue Reporting and Security

For reporting bugs, issues, or security vulnerabilities, please visit community.foss.global/. This is the central community hub for all issue reporting. Developers who sign and comply with our contribution agreement and go through identification can also get a code.foss.global/ account to submit Pull Requests directly.

□□ What It Does

`@serve.zone/catalog` provides **30+ production-ready web components** covering every aspect of server management:

- □□ **Dashboard** — Real-time cluster overview, resource usage, traffic metrics, quick actions
- □□ **Services** — Docker container management, deployment, logs, live stats, backups, and an integrated IDE workspace
- □□ **Network** — Reverse proxy configuration, DNS record management, domain & SSL certificate monitoring
- □□ **Registries** — Container registry management (onebox + external registries like Docker Hub, GHCR, ECR)
- □□ **Auth** — Login view, API token management (global + CI tokens)

- ⚙ **Settings** — Appearance, Cloudflare integration, SSL/TLS config, network settings, account management
- □□ **Platform Services** — MongoDB, MinIO, ClickHouse, Redis, Caddy monitoring and control

Every component supports **light and dark themes** out of the box and communicates via standard `CustomEvent` dispatching.

□□ Usage

Import

```
// Import everything
import * as szCatalog from '@serve.zone/catalog';

// Or import specific components
import { SzDashboardView, SzLoginView, SzServiceDetailView } from '@serve.zone/catalog';
```

Components auto-register as custom elements when imported. Use them directly in your HTML:

```
<sz-dashboard-view .data="${dashboardData}"></sz-dashboard-view>
<sz-login-view @login="${handleLogin}"></sz-login-view>
<sz-service-detail-view .service="${serviceData}" .logs="${logEntries}"></sz-service-detail-view>
```

Full Application Shell

For a complete app experience, use the demo app shell which wires up sidebar navigation, app bar menus, and all views via `dees-appui`:

```
import '@serve.zone/catalog';
// Then use <sz-demo-app-shell> for a fully configured application
```

□□ Component Reference

Dashboard

Component	Tag	Description
`SzDashboardView`	`<sz-dashboard-view>`	Main dashboard orchestrating all grid sections — cluster, services, network, infrastructure
`SzStatCard`	`<sz-stat-card>`	Single statistic card with label, value, icon, and color variant
`SzResourceUsageCard`	`<sz-resource-usage-card>`	CPU/memory progress bars, network I/O, top memory consumers
`SzTrafficCard`	`<sz-traffic-card>`	HTTP traffic metrics — requests, errors, response time, status distribution
`SzQuickActionsCard`	`<sz-quick-actions-card>`	Configurable action button grid

Dashboard Grids

Component	Tag	Description
`SzStatusGridCluster`	`<sz-status-grid-cluster>`	4-column stat card grid — total/running/stopped services, Docker status
`SzStatusGridServices`	`<sz-status-grid-services>`	Resource usage + platform services side by side
`SzStatusGridNetwork`	`<sz-status-grid-network>`	Traffic, reverse proxy, and certificates in a responsive grid
`SzStatusGridInfra`	`<sz-status-grid-infra>`	DNS/SSL status + quick actions

Services

Component	Tag	Description
`SzServicesListView`	`<sz-services-list-view>`	Table of deployed services with status badges and action buttons
`SzServiceDetailView`	`<sz-service-detail-view>`	Full service detail page — info, logs, live stats, actions, backups, and integrated workspace/IDE mode
`SzServiceCreateView`	`<sz-service-create-view>`	Service deployment form — image, ports, env vars, volumes, resource limits

Component	Tag	Description
`SzServicesBackupsView`	`<sz-services-backups-view>`	Backup schedule and backup history management

Platform Services

Component	Tag	Description
`SzPlatformServicesCard`	`<sz-platform-services-card>`	Lists infrastructure services (MongoDB, MinIO, etc.) with status indicators
`SzPlatformServiceDetailView`	`<sz-platform-service-detail-view>`	Detailed platform service view — connection info, config, metrics, logs

Network

Component	Tag	Description
`SzNetworkProxyView`	`<sz-network-proxy-view>`	Reverse proxy management — traffic targets table and access log viewer
`SzNetworkDnsView`	`<sz-network-dns-view>`	DNS record management with Cloudflare sync
`SzNetworkDomainsView`	`<sz-network-domains-view>`	Domain list with certificate status and provider info
`SzDomainDetailView`	`<sz-domain-detail-view>`	Domain detail — SSL certificate info, proxy routes, DNS records
`SzReverseProxyCard`	`<sz-reverse-proxy-card>`	Compact proxy status card (HTTP/HTTPS ports, route count)
`SzDnsSslCard`	`<sz-dns-ssl-card>`	Cloudflare DNS and ACME config status
`SzCertificatesCard`	`<sz-certificates-card>`	Certificate status counts — valid, expiring, expired

Registries

Component	Tag	Description
`SzRegistryAdvertisement`	`<sz-registry-advertisement>`	Onebox registry info card with docker quick-start commands
`SzRegistryExternalView`	`<sz-registry-external-view>`	External registry management (Docker Hub, GHCR, GCR, ECR)

Auth & Settings

Component	Tag	Description
`SzLoginView`	`<sz-login-view>`	Login page with serve.zone branding, credentials form, error display
`SzTokensView`	`<sz-tokens-view>`	API token management — global and CI tokens with copy/regenerate/delete
`SzSettingsView`	`<sz-settings-view>`	Full settings panel — appearance, Cloudflare, SSL/TLS, network, account

□□ Architecture

Component Pattern

All components follow a consistent pattern:

```
import { DeesElement, customElement, html, css, cssManager, property } from
'@design.estate/dees-element';

@customElement('sz-my-component')
export class SzMyComponent extends DeesElement {
  // TC39 standard decorators with accessor keyword
  @property({ type: String })
  public accessor label: string = '';

  public static styles = [
    cssManager.defaultStyles,
    css`
      /* Light/dark theme support */
      :host { color: ${cssManager.bdTheme('#18181b', '#fafafa')}; }
    `,
  ];

  // Events via CustomEvent (bubbles + composed for shadow DOM)
  private handleClick() {
    this.dispatchEvent(new CustomEvent('action', {
```

```
      detail: { id: this.id },
      bubbles: true,
      composed: true,
    }));
  }
}
```

Project Structure

```
@serve.zone/catalog/
├── html/                    # WccTools dev server entry point
│   ├── index.html           # HTML shell
│   └── index.ts             # WccTools config (pages + elements sections)
├── ts_web/
│   ├── index.ts             # Barrel export
│   ├── elements/            # All web components
│   │   ├── index.ts         # Element barrel export
│   │   ├── sz-*.ts          # Individual components
│   │   └── sz-demo-view-*.ts  # Demo orchestration wrappers
│   └── pages/               # Page-level components
│       ├── sz-demo-app-shell.ts  # Full app shell (dees-appui)
│       └── ...
└── dist_ts_web/             # Compiled output (npm entry point)
```

TypeScript Interfaces

The library exports comprehensive TypeScript interfaces for all data structures:

```
// Dashboard
import type { IDashboardData, IResourceUsage, ITrafficData, IClusterStats } from
'@serve.zone/catalog';

// Services
import type { IServiceDetail, IServiceStats, ILogEntry, IServiceBackup } from
'@serve.zone/catalog';

// Network
```

```
import type { IDomainDetail, ICertificateDetail, IDnsRecord, ITrafficTarget } from
'@serve.zone/catalog';

// Settings & Auth
import type { ISettings, IToken, IExternalRegistry } from '@serve.zone/catalog';
```

□□ Development

```
# Install dependencies
pnpm install

# Start dev server (wcctools dashboard with live reload)
pnpm run watch

# Production build
pnpm run build

# Run tests
pnpm test
```

The **wcctools dev server** provides an interactive dashboard where every component is rendered with demo data. Components are organized by group (Dashboard, Services, Network, etc.) in the sidebar. Demo view wrappers (`sz-demo-view-*`) are filtered out from the element list — they serve as full-page orchestration demos accessible through the Pages section.

License and Legal Information

Trademarks

This project is owned and maintained by Task Venture Capital GmbH. The names and logos associated with Task Venture Capital GmbH and any related products or services are trademarks of Task Venture Capital GmbH or third parties, and are not included within the scope of the MIT license granted herein.

Use of these trademarks must comply with Task Venture Capital GmbH's Trademark Guidelines or the guidelines of the respective third-party owners, and any usage must be approved in writing. Third-party trademarks used herein are the property of their respective owners and used only in a descriptive manner, e.g. for an implementation of an API or similar.

Company Information

Task Venture Capital GmbH Registered at District Court Bremen HRB 35230 HB, Germany

For any legal inquiries or further information, please contact us via email at hello@task.vc.

By using this repository, you acknowledge that you have read this section, agree to comply with its terms, and understand that the licensing of the code does not imply endorsement by Task Venture Capital GmbH of any derivative works.

Revision #3
Created 2026-03-28 11:14:40 UTC by foss.global Team
Updated 2026-03-28 12:21:26 UTC by foss.global Team