SamlService

September 4, 2026

SamlService

Kind: Service

Source: `atloria-monorepo/apps/api/src/auth/saml/saml.service.ts`

`SamlService` centralizes SAML service-provider configuration and authentication endpoint construction for the API. It provides URLs used during SSO initiation, assertion consumer service (ACS) handling, completion/error redirects, and organization-aware SAML configuration retrieval.

Methods

Method	Signature	Returns	Description
`spEntityId`	`spEntityId(orgId: string)`	`string`	SP entityID for an org, e.g.
`acsUrl`	`acsUrl(orgId: string)`	`string`	Assertion Consumer Service (ACS) URL — where the IdP POSTs the assertion.
`metadataUrlFor`	`metadataUrlFor(orgId: string)`	`string`	Public SP metadata URL the admin hands to their IdP.
`ssoCompleteUrl`	`ssoCompleteUrl(code: string)`	`string`	Where the browser lands after a successful SSO login (one-time-code page).
`testReturnUrl`	`testReturnUrl(result: 'ok' | 'error', message: string)`	`string`	

Method	Signature	Returns	Description
`loginErrorUrl`	`loginErrorUrl(message: string)`	`string`	
`readerReturnUrl`	`readerReturnUrl(returnTo: string`	undefined, token: string, error: string)`	`Promise<string>`
`getConfig`	`getConfig(orgId: string)`	`Promise<SamlConfig`	null>`
`getConfigByOrgSlug`	`getConfigByOrgSlug(orgSlug: string)`	`Promise<(SamlConfig & { orgSlug: string })`	null>`
`getConfigForUi`	`getConfigForUi(orgId: string)`	`unknown`	UI-safe view: NEVER returns the raw certificate, only whether one is set.
`upsertConfig`	`upsertConfig(orgId: string, dto: {`		

```
  entityId?: string;
  ssoUrl: string;
  certificate?: string;
  metadataUrl?: string | null;
  metadataXml?: string | null;
  attributeMapping?: SamlAttributeMapping | null;
  defaultRole?: UserRole;
  allowedDomains?: string[];
  enforceSso?: boolean;
})` | `Promise<SamlConfig>` | Full upsert of the org's SAML config. |
```

`toggle`	`toggle(orgId: string, isEnabled: boolean)`	`Promise<SamlConfig>`	
`importMetadata`	`importMetadata(orgId: string, input: { url?: string; xml?: string })`	`Promise<ParsedIdpMetadata>`	Fetch (from URL) or accept (inline XML) IdP metadata, parse it, and persist the discovered entityId / ssoUrl / certificate.
`recordTestResult`	`recordTestResult(orgId: string, ok: boolean, message: string)`	`Promise<void>`	
`buildSamlInstance`	`buildSamlInstance(config: Pick<SamlConfig, 'ssoUrl' | 'certificate'>, orgId: string)`	`SAML`	Build a per-request
`getLoginRedirectUrl`	`getLoginRedirectUrl(orgSlug: string, mode: 'login' | 'test' | 'reader', returnTo: string, projectId: string)`	`Promise<string>`	Build the IdP redirect URL for an SP-initiated login.
`parseRelayState`	`parseRelayState(raw: unknown)`	`SamlRelayState`	Parse RelayState back into its typed payload (defensive: default to login).

`validateCallback`	`validateCallback(orgId: string, body: Record<string, string>)`	`Promise<{ profile: Profile; config: SamlConfig }>`	Validate an IdP POST-back against the org's config.
`mapProfileToClaims`	`mapProfileToClaims(profile: Profile, mapping: SamlAttributeMapping | null)`	`SamlClaims`	Resolve logical claims from a validated SAML profile using the mapping.
`resolveOrProvisionUser`	`resolveOrProvisionUser(orgId: string, config: SamlConfig, claims: SamlClaims)`	`Promise<{ user: User; jitProvisioned: boolean }>`	Find the org's existing user for this email, or JIT-provision one.
`getSpMetadata`	`getSpMetadata(orgId: string)`	`string`	Per-org SP metadata XML the admin registers with their IdP.
`discoverByEmail`	`discoverByEmail(email: string)`	`Promise<{ orgSlug: string; enforceSso: boolean }	null>`	Map an email's domain to an org that has SSO enabled (for /discover).

Dependencies

- `PrismaService`
- `RedisService`
- `ConfigService`

Where it refuses work

- `SamlService` stops the work with `NotFoundException` when `!config` — “SSO is not configured for this organization.”, in 2 places.
- `SamlService` stops the work with `BadRequestException` when `!testPassed` — “Run a successful SSO test before enforcing SSO for the organization.”.
- `SamlService` stops the work with `BadRequestException` when `!certificate` — “An IdP signing certificate is required.”.
- `SamlService` stops the work with `NotFoundException` when `!config` — “SAML is not configured for this organization.”.
- `SamlService` stops the work with `BadRequestException` when `!xml` — “Provide either a metadata URL or metadata XML.”.
- `SamlService` stops the work with `BadRequestException` when `!res.ok`.

When something fails

- `SamlService` handles failure in 6 places: it discards it silently in 3, logs it and continues in 1, turns it into a return value in 1, and lets it reach the caller in 1.

Diagram

```
sequenceDiagram
    participant Client
    participant AuthController
    participant SamlService
    participant ConfigStore as SAML Config Store
    participant IdP as Identity Provider

    Client->>AuthController: Start SAML login
    AuthController->>SamlService: getConfigByOrgSlug()
    SamlService->>ConfigStore: Load organization SAML config
    ConfigStore-->>SamlService: SamlConfig + orgSlug
    SamlService-->>AuthController: Configuration

    AuthController->>SamlService: spEntityId(), acsUrl()
    SamlService-->>AuthController: Service provider metadata values
    AuthController-->>IdP: Redirect with SAML authentication request

    IdP-->>AuthController: POST SAML assertion to ACS URL
    AuthController->>SamlService: ssoCompleteUrl() / loginErrorUrl()
    SamlService-->>AuthController: Application redirect URL
    AuthController-->>Client: Redirect after authentication result
```

Usage

```
import { Controller, Get, Param, Redirect } from '@nestjs/common';
import { SamlService } from './saml.service';

@Controller('auth/saml')
export class SamlController {
  constructor(private readonly samlService: SamlService) {}

  @Get(':orgSlug/config')
  async getSamlConfig(@Param('orgSlug') orgSlug: string) {
    const config = await this.samlService.getConfigByOrgSlug();

    if (!config || config.orgSlug !== orgSlug) {
      return { enabled: false };
    }

    return {
      enabled: true,
      entityId: this.samlService.spEntityId(),
      acsUrl: this.samlService.acsUrl(),
```

```
      metadataUrl: this.samlService.metadataUrlFor(),
    };
  }

  @Get('complete')
  @Redirect()
  completeLogin() {
    return {
      url: this.samlService.ssoCompleteUrl(),
    };
  }
}
```

AI Coding Instructions

- Use `SamlService` as the single source of truth for SAML URLs; do not duplicate endpoint or redirect URL construction in controllers or guards.
- Handle `null` results from `getConfig()` and `getConfigByOrgSlug()` explicitly, since SAML may not be configured for every organization.
- Use `getConfigForUi()` only for client-safe configuration data; avoid exposing raw SAML secrets, certificates, or provider credentials.
- Keep organization-aware SAML flows tied to `getConfigByOrgSlug()` so the selected identity provider matches the requested organization.
- Prefer the provided completion, reader-return, and error URL helpers when redirecting after SAML authentication outcomes.

Relationships

- DEPENDS_ON → `PrismaService`
- DEPENDS_ON → `RedisService`
- DEPENDS_ON → `configservice`

Referenced By

- `SamlController` (DEPENDS_ON)
- `SamlModule` (MODULE_PROVIDES)
- `SamlModule` (MODULE_EXPORTS)
- `SsoController` (DEPENDS_ON)
- `PlatformService` (DEPENDS_ON)