Security Model

General Concepts · https://namazustudios.com/docs/general-concepts/security-model/

Elements offers a proven security model with three levels of access for users. Take advantage of SSO with Facebook and Apple accounts, or native account login - all with profile management.

Elements uses a three-tiered security model through a single API. When accessing the API a user may access at one of three access levels. Briefly, these include:

- **Anonymous** - the user is not logged in or has not provided any valid access credentials.
- **User** - the user is a regular user and can access some APIs intended for the general purpose users.
- **Superuser** - the user can access full APIs which generally includes access to all records in the system.

User Access Levels (Detailed Overview)

Currently, a user may have one of the three following access levels. This applies to the User's entire account. Across all of Elements, the security model applies as follows:

1. **Anonymous** (or Unprivileged) provides access to information that is considered public. Few APIs will supply information when used without any kind of credentials. Some APIs may only return limited sets of data at this level. Anonymous is the default access level and tends to grant very little access. If the client supplies no credentials, Elements will process all requests at this level. It is possible to greatly restrict a logged-in User's access by assigning this level. This will give the user the same access level as if they were not logged in. It may, however, still allow for valid session creation.
2. **User** is the level of access for all authenticated users. This is what the average user will use when accessing your applications. When creating new accounts, the system will automatically assign users to this level of access. User is the access level for normal users. Additionally, it is the default

level for making new accounts. Only a Superuser may escalate a user's access level allowing for access to the whole system. In general, users of your application should always be assigned this level.

3. **Superuser** allows for complete control over the system. APIs may return all information requested. Admins may perform operations such editing user accounts, resetting passwords for other users in the system, or adjust the application parameters. Using the admin console requires superuser access.

Note

Important:

- Elements intercepts incoming requests as soon as possible, reads credentials information, and applies scope based on the user-supplied credentials.
- For a majority of endpoints, including those defined as cloud functions in the scripting engine, the user scope will be set before the presentation layer receives the code.
- Elements will instantiate the specific version of a Service based on the access level.

Additionally, when troubleshooting, it is often times useful to fetch the version endpoint which prints out the revision, build time, and version number by visiting this link in your local browser http://localhost:8080/api/rest/version. This information should also be visible upon logging in to the admin console.

At the time of this writing, Elements does not support full user segmentation by group and permission scheme. This is a feature slated for future releases. However, future versions will still operate against group-based access.

Signing in from your client code

Username/password sign in is the default method of authentication within Elements which we'll use the for the example below. See here for more information on OIDC/OAuth2 sign in. Regardless of the method, you will get back a session token, as described below.

Create new User

When someone signs up for Elements (via the `UserSignupApi`), they must provide a user name, a password, and an email address. This will provide you with the new User object in the response (but does not create a session token yet!), which will be needed for the next step.

Sign in with existing User

After creating the new User, or if someone has already created a User in Elements, they can retrieve a session token with just the Username and Password via the **UsernamePasswordSessionApi**. This will return a **SessionCreation** object that contains the **SessionSecret** property, which is your session token.

Once the session token has been retrieved, you can then authenticate User related requests by adding the session token to the singleton client configuration in your generated code. For example:

```
C#
Headers["Elements-SessionSecret"] = <SessionSecret>;
```

Note

There will likely be two configurations, one for Elements and one for your application. It will be necessary to add the same session secret to both of these.

Adding this will automatically add the session token to any requests that require auth.

Warning

Most requests are profile related, so we are not quite done with the login process yet.

Now that the session secret for the User has been acquired, we can either retrieve all of the Profiles for that User, or create a new Profile.

Create New Profile

Creating a new Profile will require a reference to the Application that it's being created for. See Users and Profiles for more information on the relationship structure. The **CreateProfileRequest** object used to create a new Profile contains the following properties:

- UserId (Required)
- ApplicationId (Required)
- DisplayName (Required, can be changed later)
- ImageUrl (Optional, can be changed later)

The **CreateProfile** request will give you the created Profile object if the request was successful.

Note

It is possible to create a Profile with the signup request. If you have done this, and you only ever intend for the User to have one Profile for your Application, then it's perfectly reasonable to cache the profileId somewhere and reuse it when assigning the session token to the headers. However, it's still a good idea to implement the next section as a backup in case the cache is cleared or the user signs in from a different device.

Fetch Existing Profiles

If someone has already created a Profile and simply wishes to log back in to that Profile, then you can retrieve all profiles for a User and Application using the **GetProfiles** via the **ProfilesApi**. At this point it's up to you to decide whether to let someone choose their Profile (for example, if you have a game that allows for multiple characters) or to choose for them.

Add Profile Id to Headers

Once the Profile has been acquired, to authenticate any Profile specific requests (e.g. Progress), the Profile Id can be added as a separate header:

```
C#
Headers["Elements-ProfileId"] = <Profile Id>
```

and with this, the login process is complete!

Note

Please review the Sessions > Scoping Rules documentation to ensure you've provided the proper access levels to new users.