univention
be open.

Guardian Manual 1.1

Release 1.1

Univention GmbH

Dec 22, 2023

CONTENTS:

CHAPTER

ONE

WHAT IS THE GUARDIAN?

The Guardian provides an authorization service for apps used with a UCS system. *Authorization* means confirmation of a user's access to some resource, for example the ability to modify a user's data, export data from a system, or view a web page. It is important to note that the Guardian itself only informs about the results of any authorization request. The *app* has to enforce the result of any authorization itself.

Note: The Guardian does not provide *authentication*, confirmation that a user is who they claim to be. You can use Keycloak or another service to have a user log in, i.e., authenticate, and then use the Guardian to find out what the user is allowed to do.

1.1 Guardian apps

The authorization service consists of three applications that are installed from the UCS App Center:

- *Management API*
- *Authorization API*
- *Management UI*

At a minimum, you must install the Management API and the Authorization API. The Management UI provides an optional user-friendly graphical interface for the Management API. See the chapter on *Installation* (page 7) for more information.

1.1.1 Management API

The Management API is a REST[1] interface for *app developers* to configure aspects of the Guardian that their *apps* need in order to handle authorization. Apps should run a join script[2] during installation that hits the Management API to register with the Guardian and set up any *roles*, *permissions*, and other elements that the app needs.

This API is intended for technical audiences such as app developers. For a more user-friendly interface to manage the Guardian, please use the *Management UI* (page 21).

Please read the chapter on the *Management API and Authorization API* (page 41) for more information.

[1] https://en.wikipedia.org/wiki/REST
[2] https://docs.software-univention.de/developer-reference/latest/en/join/write-join.html#join-write

1.1.2 Authorization API

The Authorization API is a REST[3] interface that allows apps to check whether a given user or other *actor* has access to a resource that the app provides. The API can answer the following questions:

1. Given a user and a *target* resource, what is the user allowed to do?
2. Given a user, a *target* resource, and a proposed user behavior, is the user allowed to do that behavior?

This API is intended only for *app developers*. There is no user-friendly interface for the Authorization API.

Please read the chapter on the *Management API and Authorization API* (page 41) for more information.

1.1.3 Management UI

The Management UI is a user-friendly web interface that allows *guardian admins* and *guardian app admins* to configure what users in their UCS system are allowed to do once an *app* has been installed.

Please read the chapter on the *Management UI* (page 21) for more information.

1.2 What does the guardian do?

Here is an example that illustrates how the Guardian works with each of the three Guardian applications:

ACME Corporation develops an application, Cake Express, which can be installed from the UCS App Center, and which allows employees to order cakes for company events. ACME Corporation wants to allow administrators of Cake Express to have some flexibility in who gets to order cakes, so they update Cake Express so it integrates with the Guardian.

Alice works for Happy Employees, Inc. as the head IT person. When she installs Cake Express on a UCS System, the join script[4] for Cake Express does the following using the *Management API*:

1. Registers Cake Express as an *app* with the Guardian, using the name `cake-express`.
2. Creates a *namespace* called `cakes` that the app will use to store its roles and permissions for managing cakes.
3. Creates a *permission* in the `cakes` namespace that the app will check when people try to order cakes, `cake-express:cakes:can-order-cake`.
4. Creates a *role* to assign to people, `cake-express:cakes:cake-orderer`.
5. Creates a role to assign to cakes, `cake-express:cakes:birthday-cake`.

At the same time the join script registers Cake Express as an app, the Guardian creates a special role to manage Cake Express, `cake-express:default:app-admin`. Alice thinks that managing Cake Express in the Guardian should be done by an HR person, so she assigns the `cake-express:default:app-admin` role to the HR Manager, Bob, in UDM.

Bob can now log into the *Management UI*, where he is allowed to see and edit everything related to Cake Express in the Guardian. He decides to create two *capabilities*:

- Everyone in the HR department has the role `happy-employees:departments:hr`, so everyone with this role gets the permission `cake-express:cakes:can-order-cake`.
- For everyone not in the HR department, but who has the role `cake-express:cakes:cake-orderer`, they are also allowed to order cake, but not if the cake is a birthday cake with the role `cake-express:cakes:birthday-cake`, because only HR can order birthday cakes.

Bob asks Alice to give the `cake-express:cakes:cake-orderer` role to Carla, the CEO, in UDM. Now Carla is allowed to order a cake, even though she's not in the HR department.

[3] https://en.wikipedia.org/wiki/REST
[4] https://docs.software-univention.de/developer-reference/latest/en/join/write-join.html#join-write

Carla then logs into Cake Express, where she tries to order an anniversary cake for Daniel, who has been at the company for 20 years. Cake Express sends information about Carla, including her role and the name of her department and the type of cake, to the *Authorization API* to ask if she has the permission `cake-express:cakes:can-order-cake`. The Authorization API checks the capability that Bob created and determines that yes, Carla has the `cake-express:cake:cake-orderer` role and the cake is not a birthday cake, so she is allowed to order a cake.

1.3 Terminology

This section covers some of the terminology used by the Guardian in more detail.

1.3.1 Guardian admin and Guardian app admin

Guardian admins and *guardian app admins* are the two kinds of people who can manage the Guardian.

Note: Technical Note

A guardian admin has the *role* `guardian:builtin:super-admin`. This means that in UCS applications that have UDM integration, the user should have the `guardianRole` attribute include this string, i.e., `guardianRole=guardian:builtin:super-admin`.

Guardian admins can manage all aspects of the Guardian and integrated apps, including:

- *Apps*
- *Namespaces*
- *Roles*
- *Permissions*
- *Conditions*
- *Capabilities*
- *Contexts*

A guardian app admin has the ability to manage a single app that integrates with the Guardian.

Note: Technical Note

The *role* for an app admin comes in the format `<app-name>:default:app-admin`, with the `<app-name>` replaced by the unique identifier for the app. In our Cake Express example above, the app admin for Cake express has the role `cake-express:default:app-admin`. In UCS applications that have UDM integration, the user should have the `guardianRole` attribute include this string, e.g., `guardianRole=cake-express:default:app-admin`.

App admins can manage all of the aspects of their respective app:

- *Namespaces*
- *Roles*
- *Permissions*
- *Conditions*
- *Capabilities*
- *Contexts*

Note: Even if the permissions granted by the app admin role allow for all aspects of an app to be administrated, *permissions* and *conditions* cannot be managed with the *Management UI*. These types of object are intended to be created and managed by the *apps* directly during the provisioning process. Within a UCS domain this would usually happen during the join script.

1.3.2 App

An app is an application installed from the UCS App Center, or a third-party service that integrates with a UCS system, that uses the Guardian to determine what an *actor* is allowed to do.

In order to use the Guardian, apps first must register themselves using the *Management API* and a unique identifier. For example, the Cake Express app registered itself with the identifier `cake-express`. Everything in the Guardian that is used by Cake Express will start with this identifier, such as the role `cake-express:cakes:can-order-cake`.

1.3.3 Actor

An actor is a user or machine account that wants to do something in an *app*.

In the fictitious example above, Carla the CEO is an actor who wants to order a cake in Cake Express.

The Guardian does not manage actors. It is the responsibility of the app and *app infrastructure maintainers* to manage actors.

1.3.4 Target

A target is a resource that the *actor* wants to access in an *app*.

When Carla ordered an anniversary cake from Cake Express, the anniversary cake was the target resource.

The Guardian does not manage targets. It is the responsibility of the app and *app infrastructure maintainers* to manage targets.

1.3.5 Namespace

A namespace is a convenient categorization within an *app* for all aspects of the app, such as *roles* and *permissions*.

When Cake Express ran its join script at installation time, it created a namespace, `cakes`, to store everything else it created. Later, if it wants to add some kind of user management feature, it might also add a namespace called `users`. Apps also always have the `default` namespace, which is where the `app-admin` role for an app is always located.

All objects in the guardian are namespaced. When referencing the `cake-express:cakes:cake-orderer` role in Cake Express, the namespace is the second field of the role string, `cakes`.

1.3.6 Role

A role is a string that you assign to a user, group, or other database object, in order to associate it with a *capability*, either as an *actor* or as a *target*.

In the Cake Express example, Alice could assign the role `cake-express:cakes:cake-orderer` to any person or even a machine account to let that actor order a cake. Cake Express, in its own internal database, might assign the role `cake-express:cakes:birthday-cake` to distinguish between different kinds of cakes.

A role string always follows the format `<app-name>:<namespace-name>:<role-name>`.

The Guardian does not assign roles to users or objects. Instead, an *app infrastructure maintainer* is responsible for assigning role strings to the `guardianRole` attribute in UDM, or an *app developer* must assign roles to objects in their own internal database.

1.3.7 Permission

A permission is an action that an *actor* can take in an *app*.

In Cake Express, there is a permission `cake-express:cakes:can-order-cake`, that allows a user to order a cake within the Cake Express app.

Permissions are strings that are recognized by the code in an app, and used to cause something to happen in the app. Some other examples of fictitious permissions include:

- `cake-express:orders:read-order`: Allows a user to read all orders.
- `cake-express:orders:export-orders`: Allows a user to export all orders as an excel spreadsheet.
- `cake-express:users:manage-email-notifications`: Allows a user to manage the email notifications that users receive from Cake Express.

Note: The *Management UI* does not have an interface to manage permissions. This can only be done in the Management API, and as such should only be managed by *app developers*.

While a *guardian admin* technically has the ability to create permissions, the app most likely won't recognize the permission and know what to do with it.

A permission is a required component of a *capability*.

1.3.8 Condition

A condition is a criterion under which a *permission* applies.

Cake Express might have a permission `cake-express:cakes:can-add-candles` that only applies if the condition is met that the cake has the role `cake-express:default:birthday-cake`.

Note: The *Management UI* does not have an interface to manage conditions. This can only be done in the Management API, and *app developers* are most likely to manage them.

While a *guardian admin* technically has the ability to create conditions, this requires knowledge of how to write Rego[5] code.

A condition is an optional component of a *capability*.

[5] https://www.openpolicyagent.org/docs/latest/policy-language/

1.3.9 Capability

Capabilities are one of the more complicated aspects of the Guardian to explain, but they are also the key to how the *Authorization API* can answer the question of what a user or other *actor* is allowed to do in an *app*.

A capability is one or more *permissions*, optionally combined with one or more *conditions* that modify when the permission applies. A capability is then assigned to a *role* to determine what an actor with that role is allowed to do.

The simplest capability consists of a single permission. In the Cake Express example, everyone with the `happy-employees:department:hr` role is assigned a capability with a single permission, `cake-express:cakes:can-order-cake`.

A more complex capability might include a permission plus a condition. In the Cake Express example, everyone with the `cake-express:cakes:cake-orderer` role has the permission `cake-express:cakes:can-order-cake`, provided the condition that the *target* cake does not have the role `cake-express:cakes:birthday-cake`.

If there is more than one condition, the conditions are joined by a relation, either *AND* or *OR*. With *AND*, all conditions must apply: the user gets permissions if the target does not have the birthday cake role *AND* the target cake is not marked as a "top-tier" style cake. With *OR*, any condition can apply: the user gets permissions if they made the cake order *OR* the cake is an anniversary cake.

1.3.10 Context

A context is an additional tag that can be applied to a *role*, to make it only apply in certain circumstances.

For example, Happy Employees, Inc. has two different offices, London and Berlin. They have the party-planner role, and Daniel is the party-planner for London and Erik is the party-planner for Berlin. ACME sets up a *capability* that says that a party-planner can order a cake, but only for the office context where they are a party-planner. So Erik can't order a cake for London, and Daniel can't order a cake for Berlin.

Not all *apps* support contexts. Please check with the *app developer* for your app, to see if they support contexts.

CHAPTER

TWO

INSTALLATION

2.1 Installation on a UCS primary node

The different components of the Guardian can be installed from the Univention App Center. A prerequisite for using the Guardian is a working Keycloak installation in the UCS domain. How to install and configure Keycloak in a UCS domain can be found here[6].

To install all required components on a UCS primary node, run:

```
univention-app install \
    guardian-management-api \
    guardian-authorization-api \
    guardian-management-ui
```

Most of the settings are configured automatically, but since this is a preview version, some manual configuration steps remain.

`KEYCLOAK_SECRET` can be obtained by running the following command:

```
KEYCLOAK_SECRET=$(univention-keycloak oidc/rp secret --client-name guardian-cli |␣
↪sed -n 2p | sed "s/.*'value': '\([[:alnum:]]*\)'.*/\1/")
```

Update settings for the *Management UI*:

```
univention-app configure guardian-management-api --set \
    "guardian-management-api/oauth/keycloak-client-secret=$KEYCLOAK_SECRET"
```

Then set your `USERNAME` and `PASSWORD` to credentials for a user which has access to the UDM REST API:

```
USERNAME=Administrator
PASSWORD=password
```

Then update settings for the Guardian *Authorization API*:

```
univention-app configure guardian-authorization-api --set \
    "guardian-authorization-api/udm_data/username=$USERNAME" \
    "guardian-authorization-api/udm_data/password=$PASSWORD"
```

To be able to use the Guardian Management UI, it is also necessary to give the user the required permissions. For this the Management UI already utilizes the Guardian. This means the user needs to get the proper `guardianRole` assigned. To make the Administrator account the *Guardian super user*, who has all privileges, execute:

```
udm users/user modify --dn uid=Administrator,cn=users,$(ucr get ldap/base) \
    --set guardianRole=guardian:builtin:super-admin
```

With these steps the Guardian setup is complete and the Management UI should be available from the Univention Portal.

[6] https://docs.software-univention.de/keycloak-app/latest/index.html

2.1.1 Configuring Keycloak for join scripts

When installing an *app* that uses the Guardian, it will need a special Keycloak client that is configured specifically for join scripts.

Run the following command on the server with the Guardian Management API installed:

```
GUARDIAN_SERVER="$(hostname).$(ucr get domainname)"
univention-keycloak oidc/rp create \
    --name guardian-scripts \
    --app-url https://$GUARDIAN_SERVER \
    --redirect-uri "https://$GUARDIAN_SERVER/univention/guardian/*" \
    --add-audience-mapper guardian-scripts
```

Then configure the new client using the Keycloak web interface. Choose *ucs* from the realm drop-down list at the top of the left navigation bar. Then click on *Clients* in the left navigation bar, and choose *guardian-scripts*.

Configure password login for scripts and remove the client secret:

1. Go to the *Settings* tab.
2. Navigate to the *Capability config* section.
3. Turn *Client authentication* off.
4. Under *Authentication flow*, check the checkbox for *Direct access grants*.

Click the *Save* button at the bottom of the screen.

Configure the correct audience for the Guardian:

1. Go to the *Client scopes* tab.
2. Click on *guardian-scopes-dedicated*.
3. **Choose *Add mapper* ▸ *By configuration*.**
    1. Select *Audience*.
    2. For the *Name*, use `guardian-audience`.
    3. For the *Included Client Audience*, choose `guardian`.
4. **Choose *Add mapper* ▸ *By configuration*.**
    1. Select *User Attribute*.
    2. For the *Name*, use `dn`.
    3. For the *User Attribute*, use `LDAP_ENTRY_DN`.
    4. For the *Token Claim Name*, use `dn`.
    5. Turn *Add to ID Token* off.
    6. Turn *Add to userinfo* off.
    7. Verify that *Add to access token* is on.

Click the *Save* button at the bottom of the screen.

2.2 Installation on different UCS server roles

This setup assumes that all Guardian components are installed on the same host and that Keycloak as well as the UDM REST API are running on that host as well. This is usually the UCS primary node. The Guardian supports the installation of its components on any UCS server role as well as distributing the individual components on different hosts. For that to work though, multiple settings regarding URLs for Keycloak, the UDM REST API and the different Guardian components themselves have to be configured manually. Please check the chapter Configuration for a full reference of all the app settings.

2.3 Load balancing and multiple instances

The Guardian was developed with the capability of running multiple instances of each component in mind. It is possible to deploy multiple instances of the Guardian Management UI and Guardian Authorization API apps in the UCS domain without any problems, as long as they are properly configured.

The Management API should only be deployed once in any UCS domain due to the limitations mentioned in *App Center database limitations* (page 59).

CHAPTER

THREE

CONFIGURATION

This chapter is a reference to all *app* settings of the Guardian divided by component. These settings can be configured either via the `univention-app` command line interface or the Univention App center dialog for app settings.

To change the log level for the *Management API* for example, use the following command:

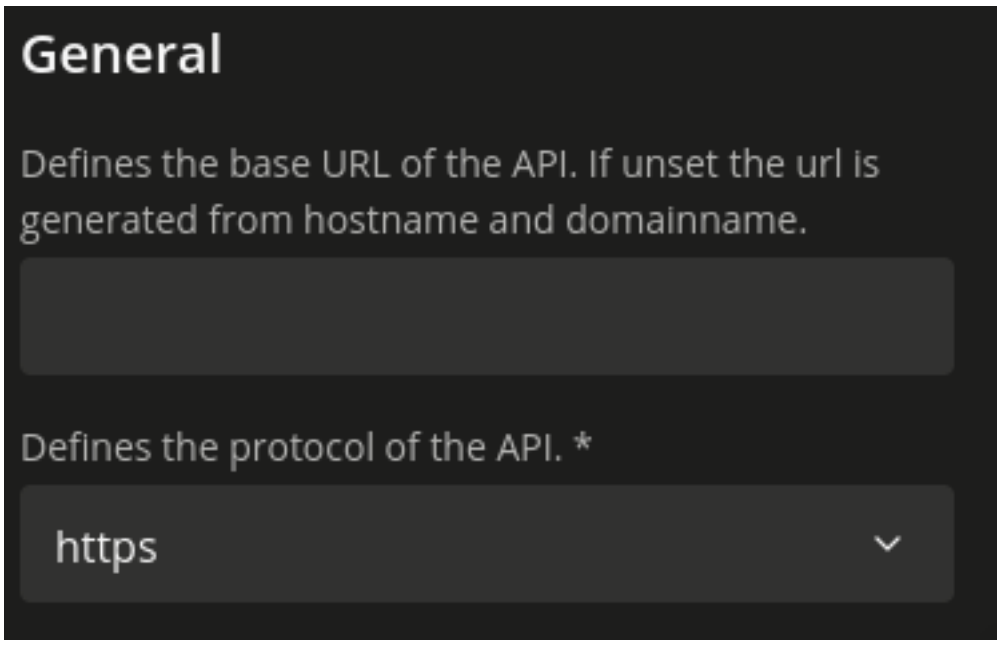
```
univention-app configure guardian-management-api --set \
   "guardian-management-api/logging/level=ERROR"
```

If any of the settings are changed, the application is restarted automatically.

3.1 Guardian Management API

3.1.1 General

guardian-management-api/base_url



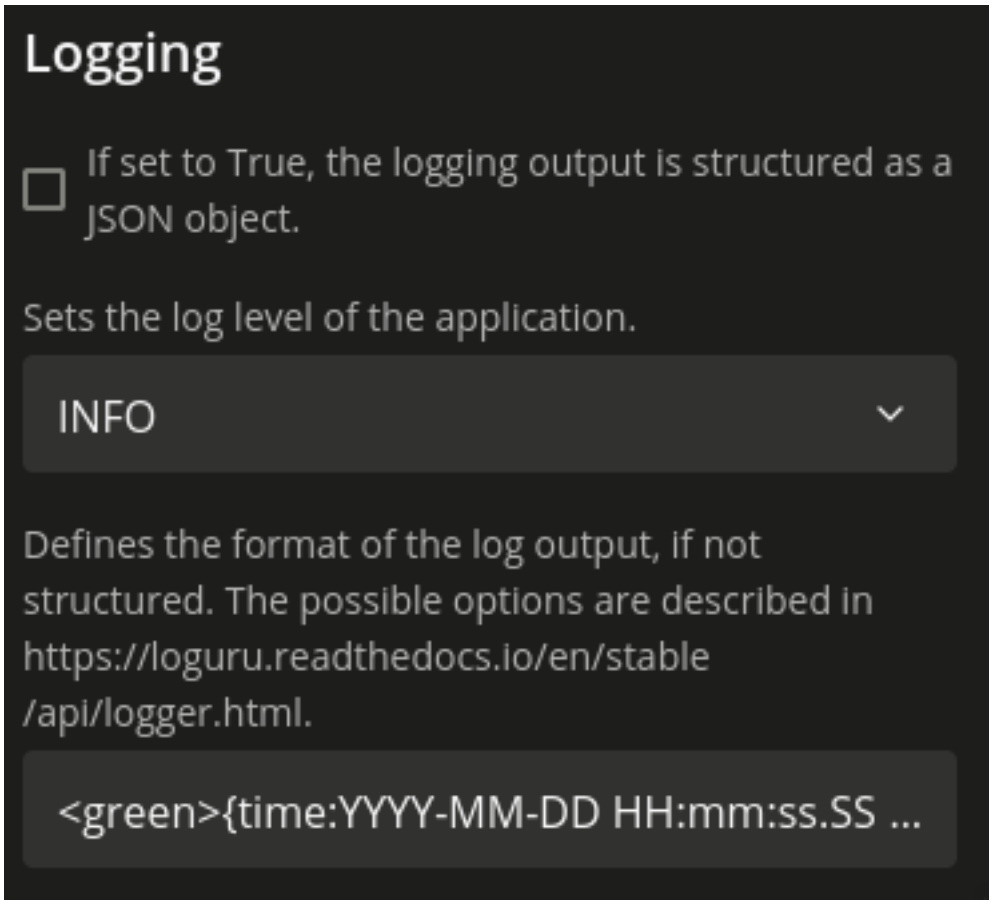
Defines the base URL of the API. If unset the URL is generated from hostname and domainname of the server the API is installed on. You must not specify the protocol here as this is set in *guardian-management-api/protocol* (page 11).

guardian-management-api/protocol

Defines the protocol of the API. Can be either `http` or `https`. Default is `https`.

3.1.2 Logging

guardian-management-api/logging/structured



Can be either `True` or `False`. If set to `True`, the logging output of the Management API is structured as json data.

guardian-management-api/logging/level

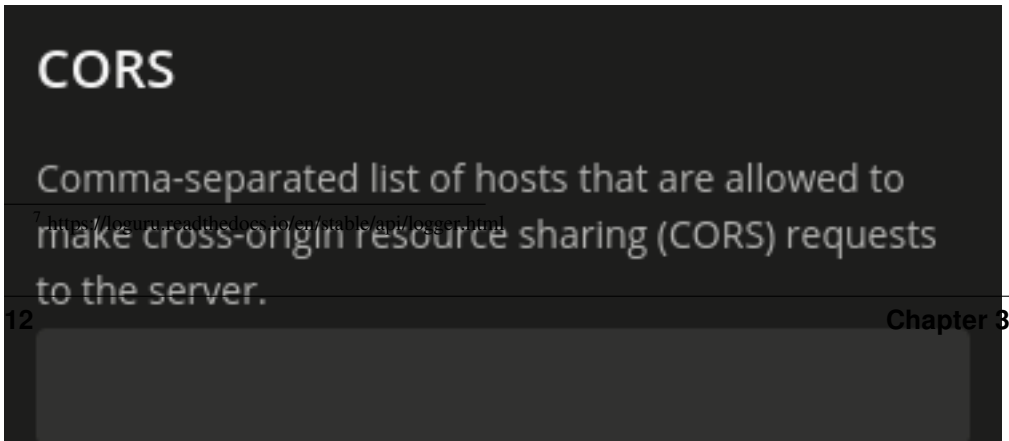
Sets the log level of the application. It can be one of `DEBUG`, `INFO`, `WARNING`, `ERROR`, `CRITICAL`.

guardian-management-api/logging/format

This setting defines the format of the log output if *guardian-management-api/logging/structured* (page 12) is set to `False`. The documentation for configuring the log format can be found here[7].

3.1.3 CORS

guardian-management-api/cors/allowed-origins



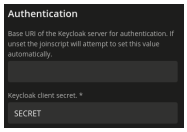
Comma-separated list of hosts that are allowed to make cross-origin resource sharing (CORS) requests

[7] https://loguru.readthedocs.io/en/stable/api/logger.html

to the server. At a minimum, this must include the host of the *Management UI*, if installed on a different server.

3.1.4 Authentication

guardian-management-api/oauth/keycloak-uri



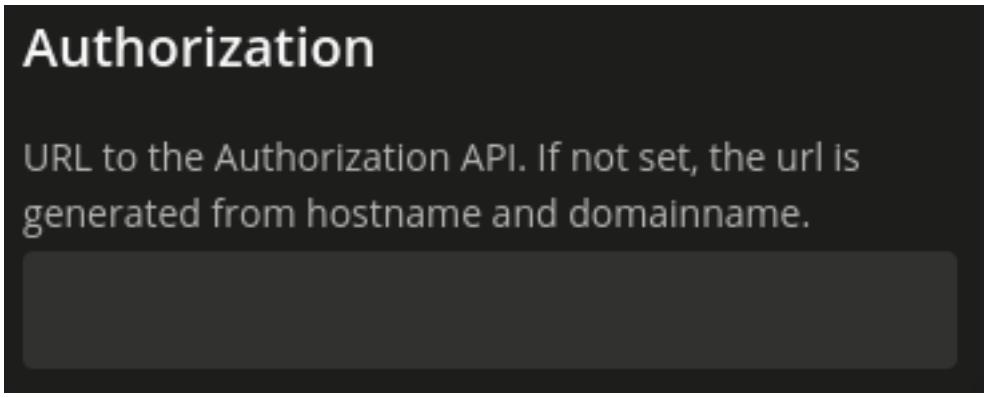
Base URI of the Keycloak server for authentication. If unset the application tries to derive the Keycloak URI from the UCR variable `keycloak/server/sso/fqdn` or fall back to the domain name of the host the application is installed on.

guardian-management-api/oauth/keycloak-client-secret

Keycloak client secret.

3.1.5 Authorization

guardian-management-api/authorization_api_url



URL to the Authorization API. If not set, the URL is generated from hostname and domain name of the server the application is installed on.

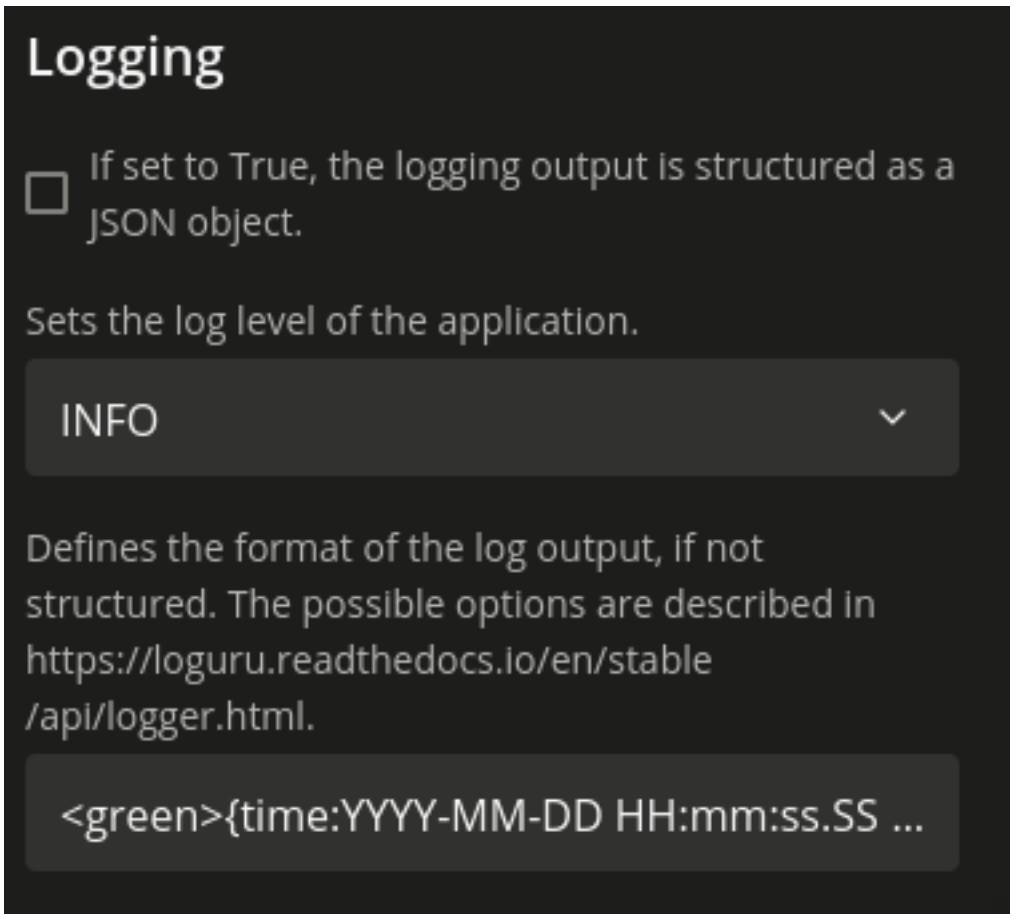
3.2 Guardian Authorization API

guardian-authorization-api/bundle_server_url

URL to the Management API from which to fetch the policy data for decision making. If not set, the URL is generated from hostname and domain name of the server the application is installed on.

3.2.1 Logging

guardian-authorization-api/logging/structured



Can be either `True` or `False`. If set to `True`, the logging output of the Authorization API is structured as json data.

guardian-authorization-api/logging/level

Sets the log level of the application. It can be one of `DEBUG`, `INFO`, `WARNING`, `ERROR`, `CRITICAL`.

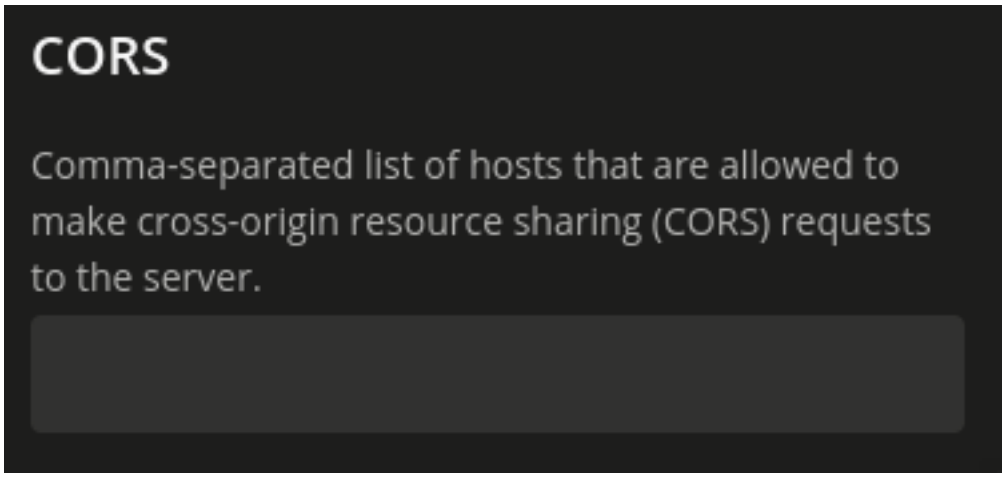
guardian-authorization-api/logging/format

This setting defines the format of the log output if *guardian-authorization-api/logging/structured* (page 14) is set to `False`. The documentation for configuring the log format can be found here[8].

[8] https://loguru.readthedocs.io/en/stable/api/logger.html

3.2.2 CORS

guardian-authorization-api/cors/allowed-origins



Comma-separated list of hosts that are allowed to make cross-origin resource sharing (CORS) requests to the server. You may need to add third-party *apps* to this list, if they need to use the Guardian.

3.2.3 UDM

guardian-authorization-api/udm_data/url

The URL of the UDM REST API for data queries.

guardian-authorization-api/udm_data/username

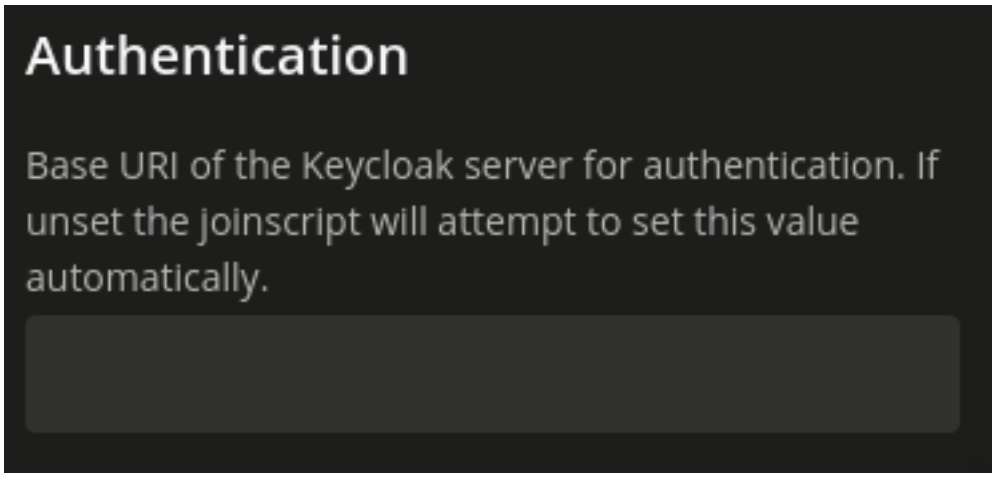
Username for authentication against the UDM REST API.

guardian-authorization-api/udm_data/password

Password for authentication against the UDM REST API.

3.2.4 Authentication

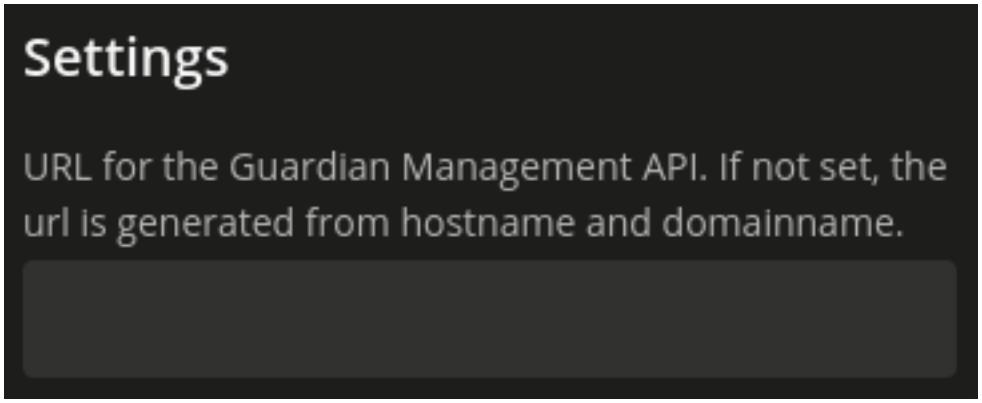
guardian-authorization-api/oauth/keycloak-uri



Base URI of the Keycloak server for authentication. If unset the application tries to derive the Keycloak URI from the UCR variable `keycloak/server/sso/fqdn` or fall back to the domain name of the host the application is installed on.

3.3 Guardian Management UI

guardian-management-ui/management-api-url

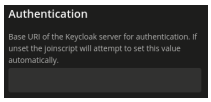


URL for the Guardian Management API. If not set, the URL is generated from hostname and domain name.

3.3.1 Authentication

guardian-management-ui/oauth/keycloak-uri

Base URI of the Keycloak server for authentication. If unset the application tries to derive the Keycloak URI from the UCR variable `keycloak/server/sso/fqdn` or fall back to the domain name of the host the application is installed on.



CHAPTER

FOUR

TROUBLESHOOTING

4.1 Introduction

This chapter provides a guide to troubleshooting the Univention Guardian. It assumes that you have a basic understanding of the Guardian and its components, as well as familiarity with command-line tools like Docker.

4.2 Common issues

Before attempting any other solutions, please follow these steps:

1. Restart all Guardian services.
2. Check connectivity between the Guardian components.

Here are some examples of how to do this:

- Connectivity *Management API* -> *Authorization API*:

Listing 4.1: Check the connectivity between the Management API and the Authorization API

```
univention-app shell guardian-authorization-api
apt update; apt install -y curl
curl $GUARDIAN__MANAGEMENT__ADAPTER__AUTHORIZATION_API_URL/openapi.json -I
# check for 200 OK
```

- Connectivity OPA -> Management API:

Listing 4.2: Check the connectivity between the OPA and the Management API

```
univention-app shell -s opa guardian-authorization-api
apt update; apt install -y curl
# check connection to management API
curl -I $OPA_GUARDIAN_MANAGEMENT_URL/openapi.json # check for 200 OK
# check if the bundle can be retrieved
curl -I $OPA_GUARDIAN_MANAGEMENT_URL/$OPA_POLICY_BUNDLE # check for 200 OK
```

- Connectivity UI -> Management API: Use the developer tools in your browser to check the network tab for errors.

If any of these steps fail, there could be several reasons:

1. The Guardian component that you're trying to reach might not be running or couldn't start properly. Check the logs of the component and restart it if necessary.
2. The Guardian component that you're trying to reach is running but not reachable from the component you're currently on. This could be due to faulty configuration or connectivity problems. Check the environment

variables inside the container with the command `env` and check the connectivity between the containers with the command `ping`.

3. The Guardian component that you're trying to reach is running and reachable but doesn't respond to the request. This could be the case if the Management API is indeed running but the OPA bundle is not generated. Check the logs of the component and restart it if necessary.

4.3 First time installation and configuration

Make sure that you complete all steps of the *configuration* (page 11) process. Services might not work properly if the configuration is not complete.

4.4 Management UI

If the Guardian UI loads but with an error, check the network and console tabs of your browser's developer tools. There you can see if the UI could connect to the Management API and if the Management API responded with an error. If the Management API responded with an error, check the logs of the Management API.

4.5 Management API

4.5.1 Not authorized to access the Authorization API

If in the Management API logs you see the following error: `ERROR | Unsuccessful response from the Authorization API: {'Detail': 'Not Authorized'}`, then the Management API could not authorize itself to the Authorization API. For more information, check the logs of the Authorization API. This can happen if the client secret is not configured for the Management API or is wrong.

4.6 Debugging OPA decisions

The OPA decisions can't be easily debugged at the moment. However, there are some ways to make sure everything is working as expected:

1. OPA fetches the bundle from the Management API. The bundle contains the policies and the data that OPA needs to make decisions. The bundle is generated by the Management API from its database.
2. If OPA cannot fetch the bundle, it will show it in its logs. Whenever there's an update in the Management API, the bundle is regenerated and OPA will fetch it again and log it.
3. To inspect the contents of the bundle, use the following commands:

Listing 4.3: Inspect OPA bundle contents

```
univention-app shell guardian-management-api
apt update; apt install -y jq
jq '.' /guardian_service_dir/bundle_server/build/GuardianDataBundle/guardian/
↪mapping/data.json
```

There you can see what permissions get assigned to which roles under which conditions.

4.7 Authentication issues

If you cannot log in to the Guardian UI or to any of the Swagger UIs for the Management API or the Authorization API, make sure that the Keycloak server is reachable. You can check the logs of the Keycloak container with the following command:

Listing 4.4: Check Keycloak logs

```
univention-app logs keycloak
```

The most common issues are invalid redirect URLs and clock issues.

For the redirect URL, make sure that the URL is correct. You can check the configuration of the Keycloak server at the following URL: `https://ucs-sso-ng.school.test/admin/master/console/#/ucs/clients`. Make sure that the redirect URL matches the URL of the Guardian UI for the `guardian-ui` client, including the scheme (e.g., `https://`).

For clock issues, a small difference between the clock of the Keycloak server and the clock of the Management API or the Authorization API can cause authentication issues. If this is the case, you will see it in the logs of the Management API or the Authorization API. Look for: `WARNING | Invalid Token: "The token is not yet valid (iat)"`.

CHAPTER

FIVE

MANAGEMENT UI

This chapter is geared towards *Guardian administrators* who want to manage *roles* and related objects which can grant *permissions* to users.

The Guardian *Management UI* app provides a web interface to manage some of the features of the REST API of the Guardian *Management API* app. The following sections describe which functions can be performed with the web interface.

You can access the **Guardian Management UI** under `https://[Domainname]/univention/guardian/management-ui` for the `Domainname` where the **Guardian Management UI** app is installed. When installing the app, a portal entry is created in the *Administration* category of the default domain portal (`cn=domain,cn=portal,cn=portals,cn=univention,$ldap_base`). With the default configuration, a user who wants to use the **Guardian Management UI** as a *guardian admin* needs the role `guardian:builtin:super-admin`.

For a detailed explanation on what *roles*, *capabilities*, *namespaces* and *contexts* are, refer to the *section about terminology* (page 3).

5.1 General remarks

After you entered the **Guardian Management UI**, you will see a navigation bar with the entries *ROLES*, *NAMESPACES* and *CONTEXTS*, a search bar with filters and a table.

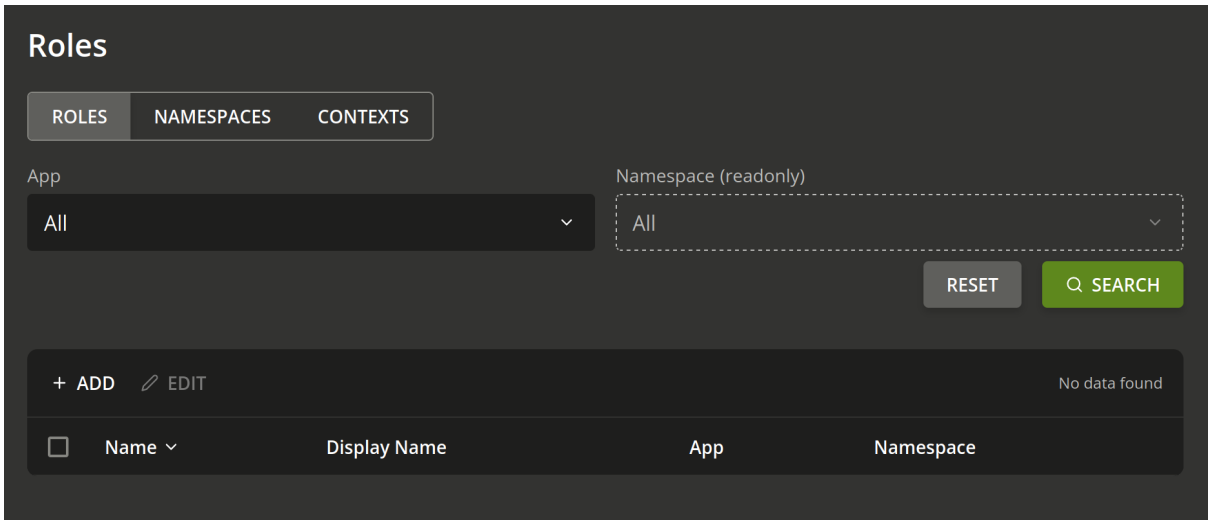


Fig. 5.1: The front page of the **Guardian Management UI**.

There are some differences, but you can view and manage the object types *role*, *namespace*, and *context* by navigating between them with the navigation bar as described in the following sections. The management of *capabilities* is done while editing a role.

Note: The *apps* in the *App* box can only be managed via the REST API provided by the **`Guardian Management API`** app. Refer to *the developer quick start documentation* (page 45) if you need to integrate an app with the Guardian.

In the search view for one of the object types, you can filter by app and namespace, with the exception of namespaces themselves, which can only be filtered by app.

Note: At the moment it is not possible to include properties of an object, such as its *Display Name*, in the search criteria.

5.2 Roles

The **`Guardian Management UI`** can be used to manage *roles*. A role contains capabilities and is defined within the scope of an app and a namespace. From the role and its capabilities, permissions are derived. For more information about the fundamental concepts, refer to the *section about terminology* (page 3).

5.2.1 Create a new role

To create a new role first open the **`Guardian Management UI`** and click on *ROLES* in the navigation menu.

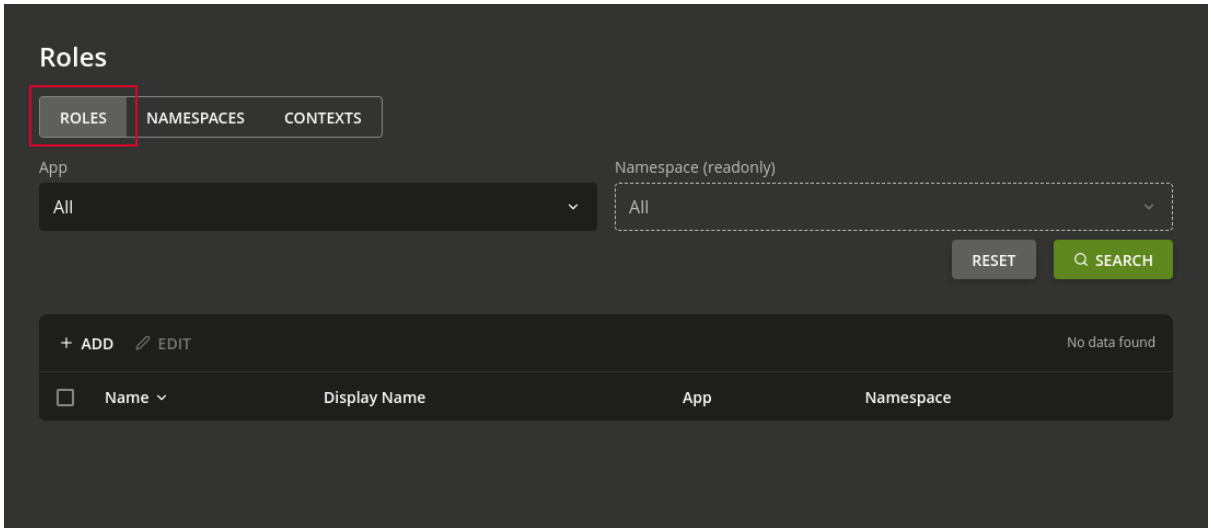


Fig. 5.2: Link to the roles page.

Then click on the + *ADD* button to open the page to create a new role.

The page to create a new role looks like this:

Fill out all the necessary fields and click on the *CREATE ROLE* button to create the role. A pop-up will be shown which confirms the creation by displaying the role name.

Note: The selectable options for the *Namespace* box depend on the selected app in the *App* box. You have to select an app first before you can select a namespace. If you selected an app and still don't see any selectable namespaces that means that there are no namespaces for that app. Refer to the *section about creating namespaces* (page 32).

Note: Capabilities for a role can only be managed on existing roles. To add capabilities to the role you are currently creating first create the role with the *CREATE ROLE* button and then manage capabilities as described in *Capabilities*

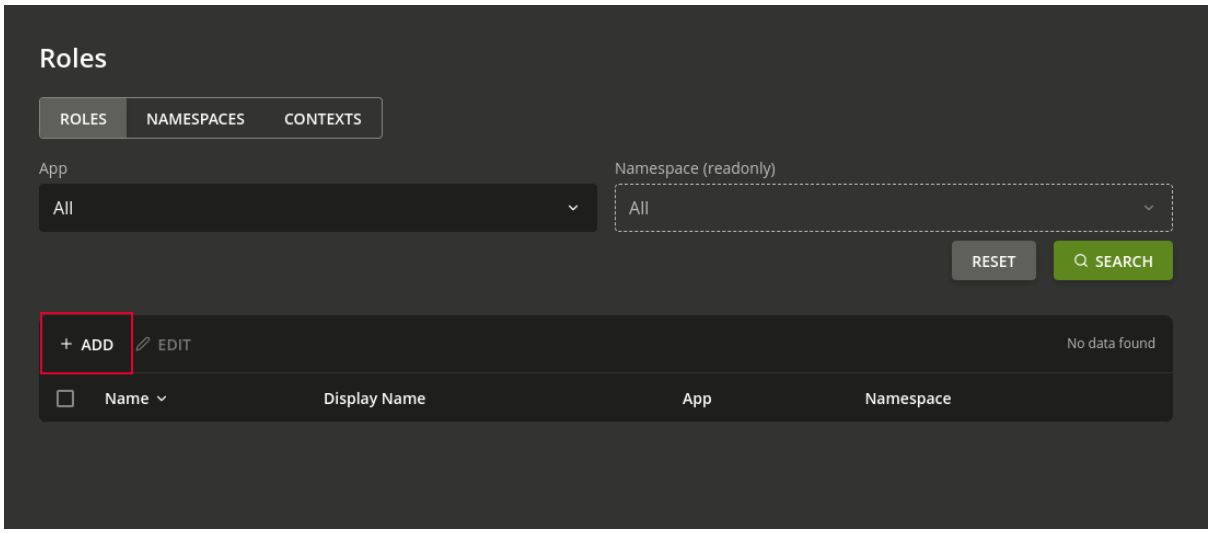


Fig. 5.3: Click + *ADD* to create a new role.

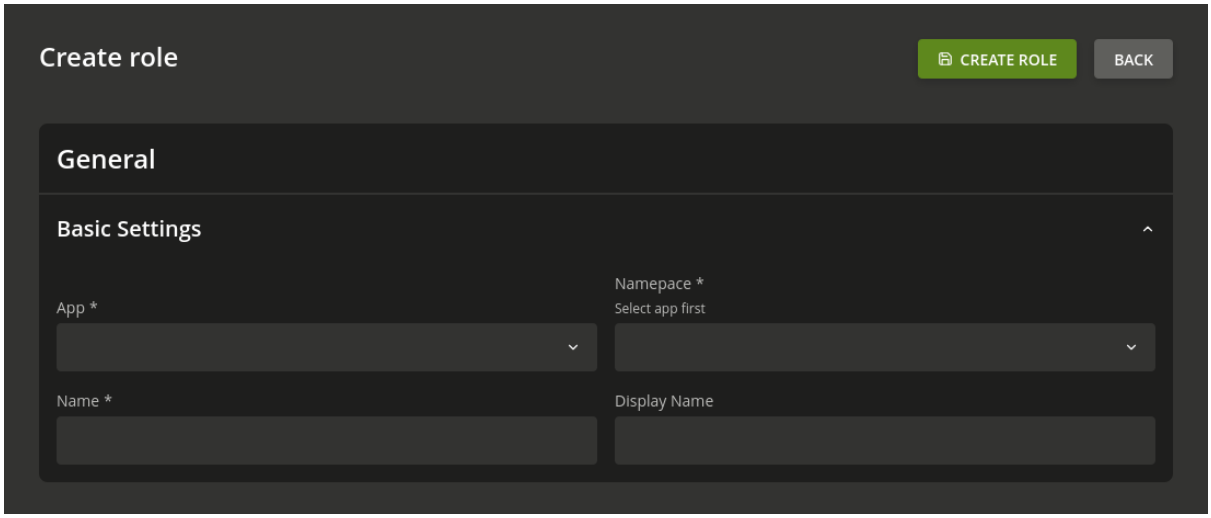


Fig. 5.4: Page to create a new role.

of a role (page 26).

5.2.2 Listing and searching roles

To list existing roles open the "Guardian Management UI" and click on *ROLES* in the navigation menu.

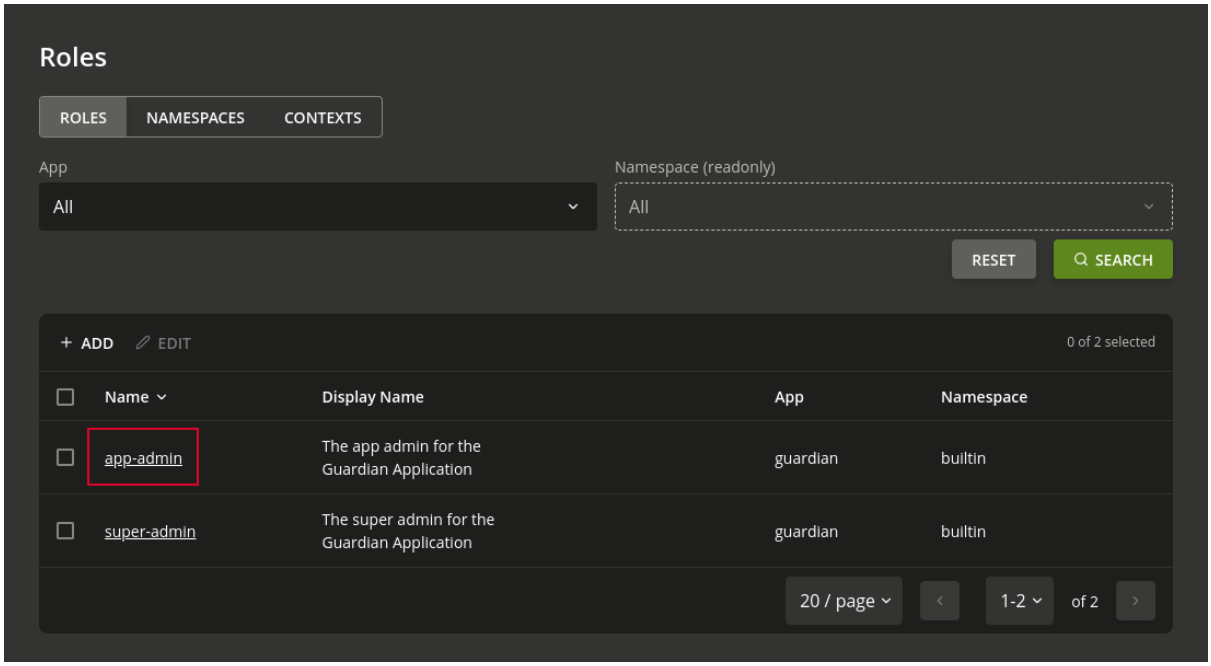


Fig. 5.5: Link to the "Roles" page.

On this page you can search for existing roles by clicking the *SEARCH* button. The results will be shown below that button. The search can be narrowed down by selecting a specific app in the *App* box, and a namespace of the selected app in the *Namespace* box.

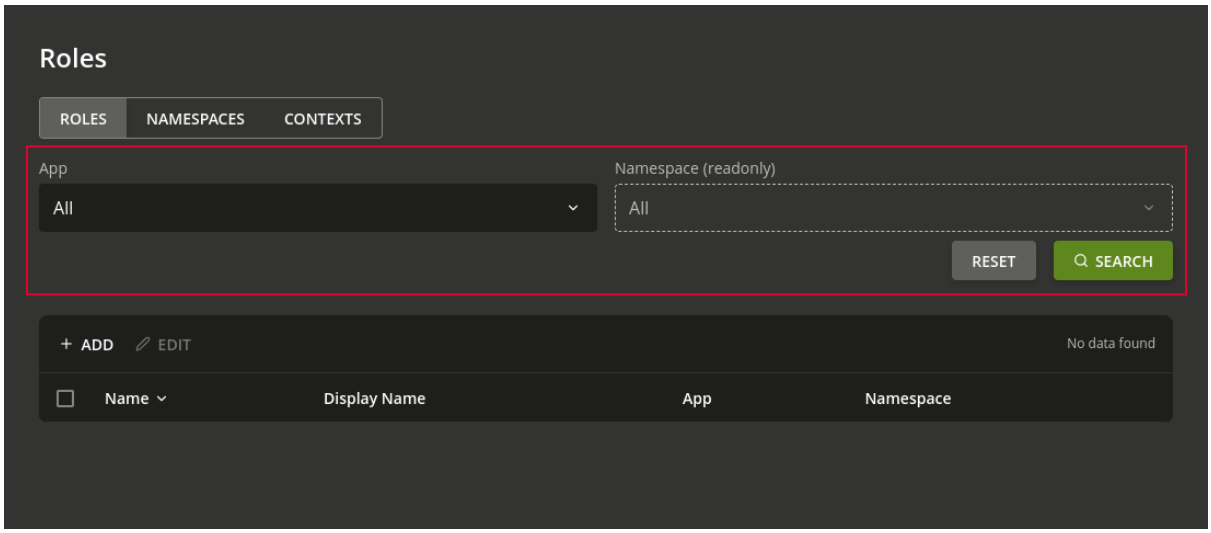


Fig. 5.6: Form elements for the search of roles.

Note: The namespaces for the *Namespace* box can be managed as described in *Namespaces* (page 32).

5.2.3 Editing existing roles

To edit a role, follow the steps in *Listing and searching roles* (page 24) to list them and then click on the name of the role you want to edit.

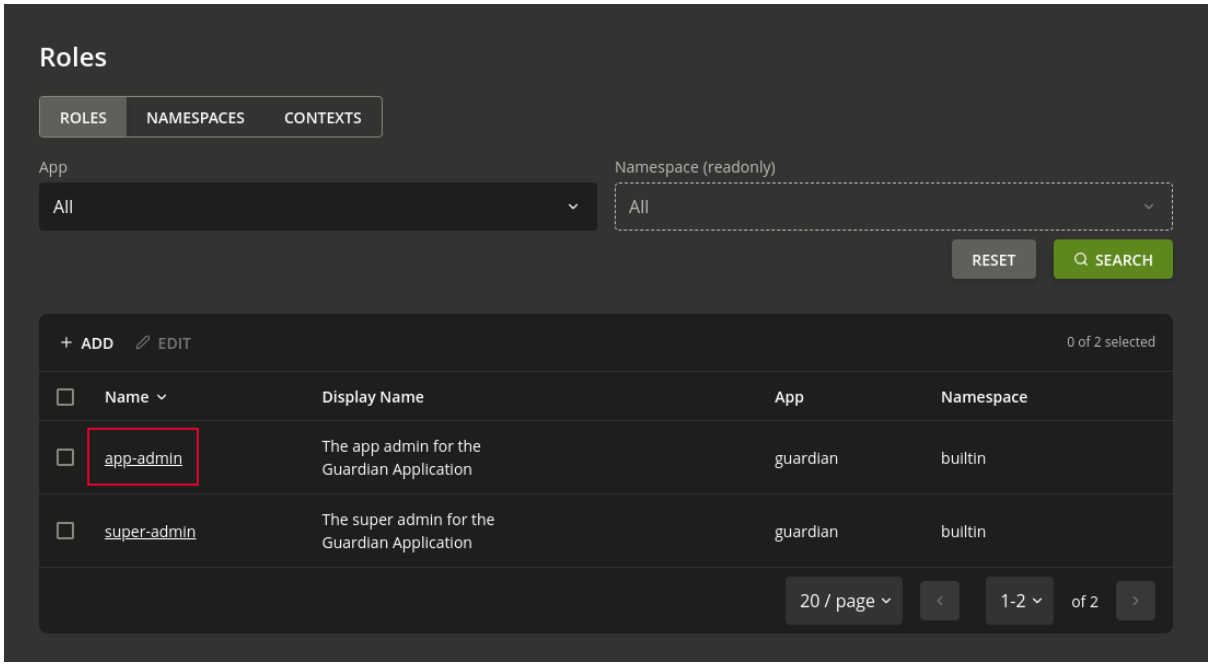


Fig. 5.7: Edit button for listed roles.

The role editing is split into two pages.

The first page is to edit the direct properties of the role and is the first page you see when opening a role. This page can be accessed by clicking *ROLE* in the navigation menu. Here you can edit the fields you want to change and click on *SAVE* to save the changes.

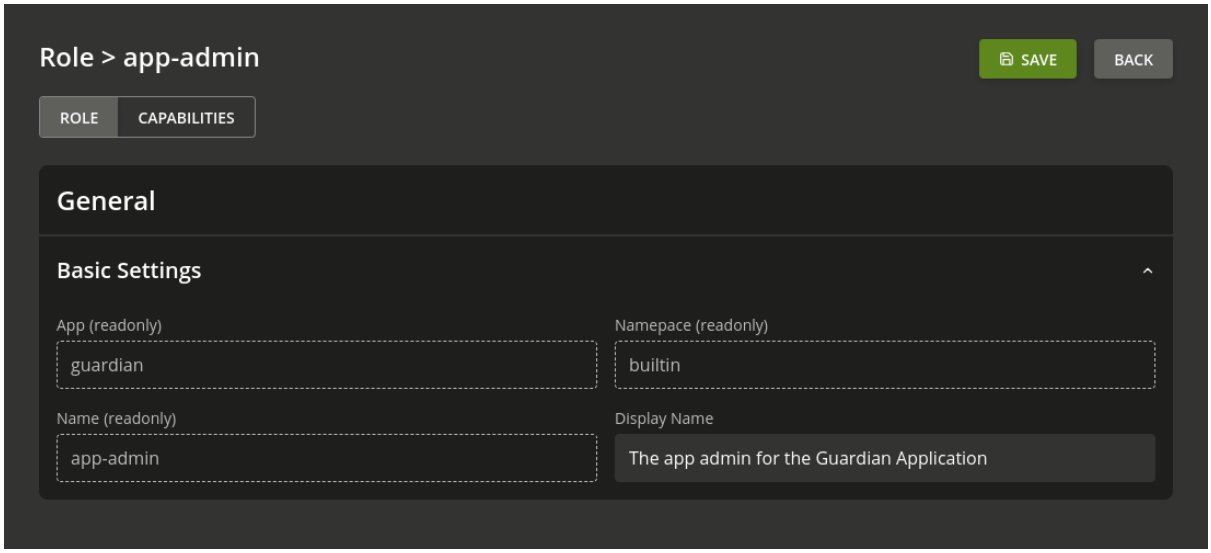


Fig. 5.8: View and edit page of an existing role.

The second page is to manage the capabilities of the current role. This page can be visited by clicking on *CAPABILITES* in the navigation menu.

Here you can list all capabilities of the role you are currently editing and manage them. You can also create new capabilities for that role or delete existing ones. For more details on capabilities see the section: *Capabilities of a role*

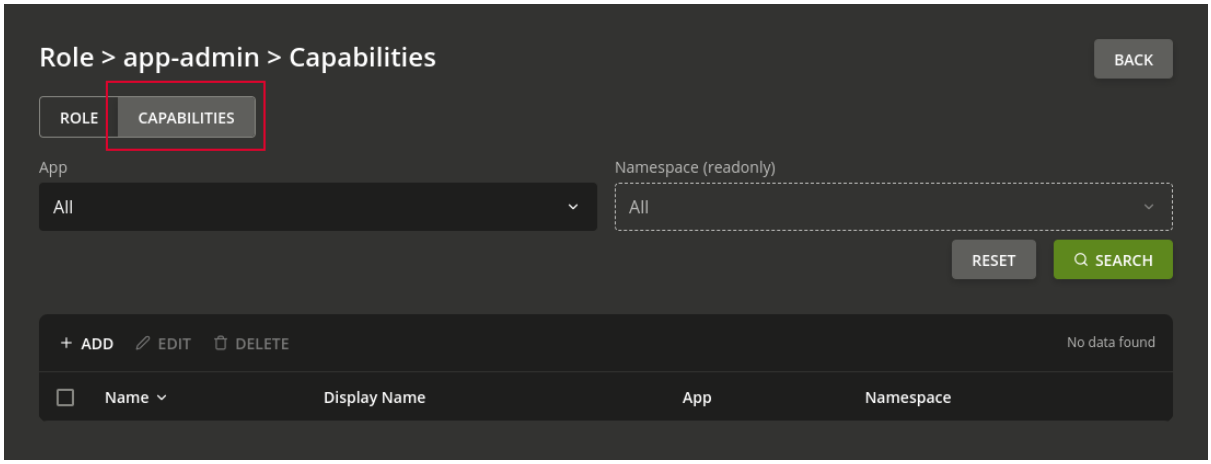


Fig. 5.9: Link to the "Capabilities" page of an existing role.

(page 26).

5.2.4 Deleting roles

Deleting roles is not possible at the moment. Neither through the web-interface nor the REST API.

5.3 Capabilities of a role

Capabilities serve as the means to manage the *permissions* the *role* will grant to the user it is attached to.

Each capability object can define one ore more permissions it will grant. These permissions can only be selected for a specific app and namespace. If you want to grant permissions for different apps and/or namespaces you have to create multiple capability objects.

Inside an capability object you can also add *conditions* that influence whether the permissions are actually granted.

The capabilities work on a whitelist principle and do not collide.

Note: Capabilities can only be managed on existing roles.

If you are creating a new role and want to manage its capabilities, first create the role and then edit the role to manage its capabilities.

5.3.1 Create new capability for a role

To add a capability for a role, first click on *CAPABILITES* in the navigation menu while editing a role. See *Editing existing roles* (page 25) for more details on editing a role.

Then click on the + *ADD* button to open the page to create a new capability.

The page to create a new capability looks like this:

To create the capability fill out all the necessary fields and then click the *CREATE CAPABILITY* button. A pop-up will be shown which confirms the creation by displaying the capability name.

Three noteworthy fields are the list of *Permissions*, the list of *Conditions* and the *Relation*.

Permissions

In the *Permissions* list you can edit all permissions the capability will grant if the conditions in the *Conditions*

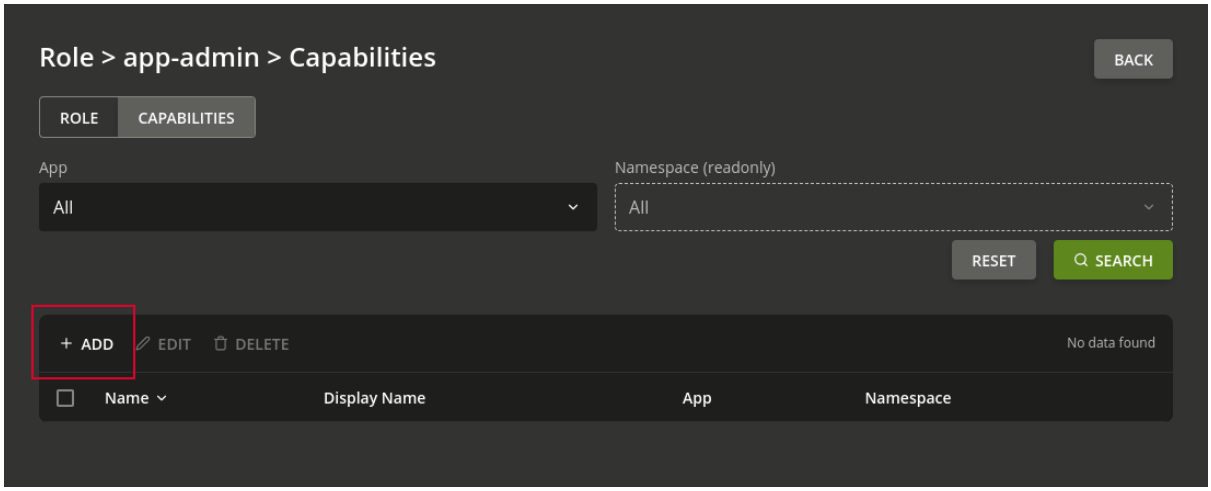


Fig. 5.10: Click + *ADD* to create a new capability.

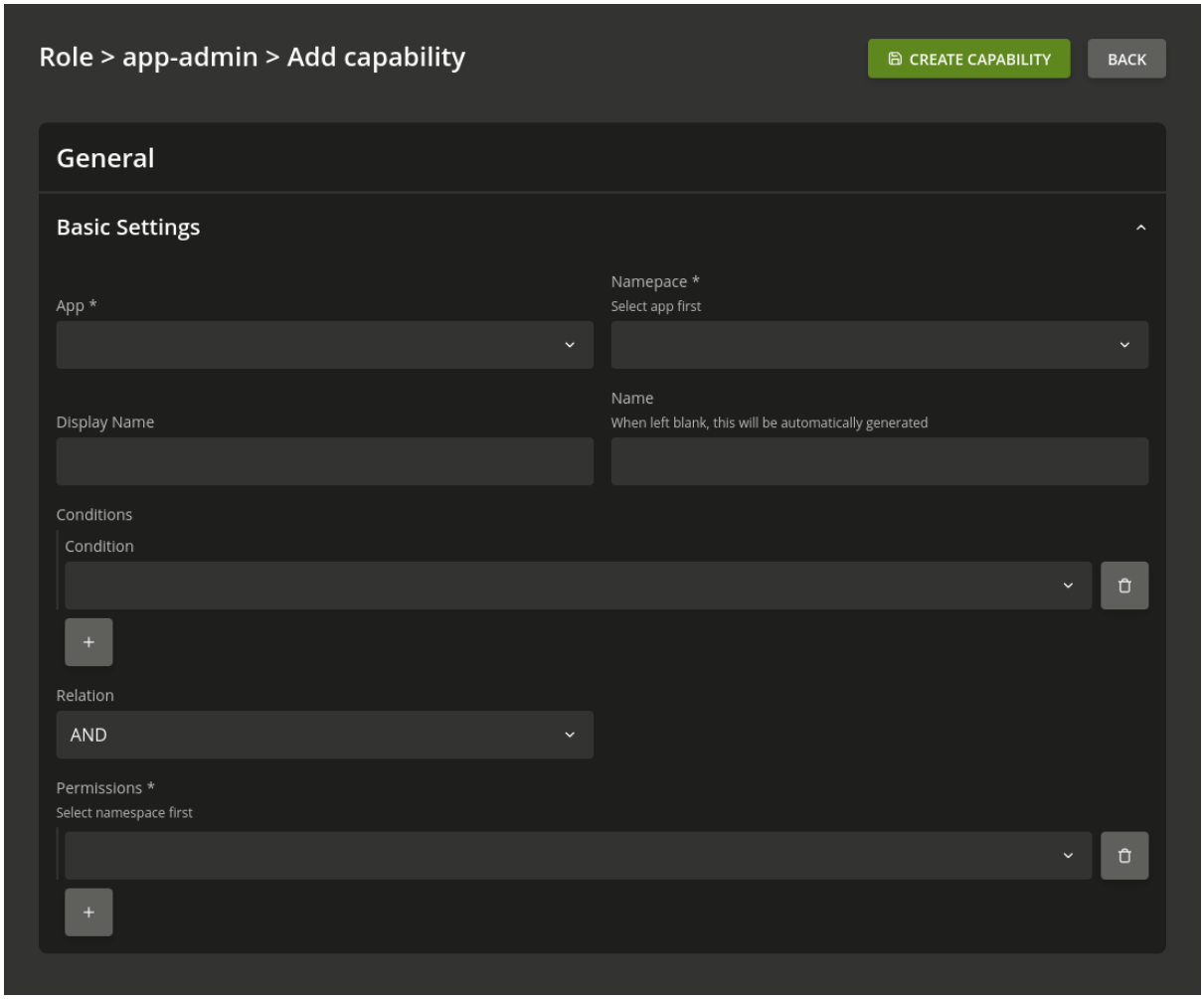


Fig. 5.11: Page to create a new capability.

list are met. The available permissions are based on the selected app in the *App* box and namespace in the *Namespace* box. You cannot select any permissions before filling out both of these fields.

Note: If both the *App* box and *Namespace* box are filled out, and you still cannot select permissions, this means that no permissions exist for that app and namespace.

Conditions

In the *Conditions* list you can edit all the conditions that should be checked before the permissions in the *Permissions* list are granted. Some conditions require additional parameters. You can look up more about these conditions in chapter *Conditions Reference* (page 61). Additional fields will be shown underneath them once selected.

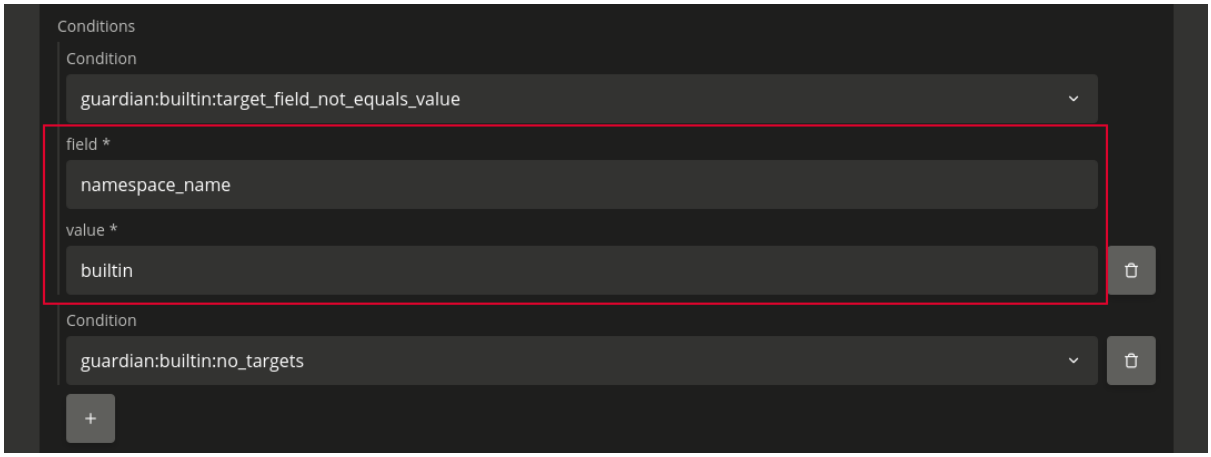


Fig. 5.12: Condition with extra parameters.

Note: See *Conditions Reference* (page 61) for an explanation of the pre-existing conditions.

Relation

The value of the *Relation* box describes how the **`Guardian Authorization API`** will check conditions during authorization. *AND* means all conditions must be met and *OR* means only 1 condition must be met.

5.3.2 Listing and searching capabilities of a role

To list capabilities of a role click on *CAPABILITES* in the navigation menu while editing a role. See *Editing existing roles* (page 25) for more details on editing a role.

On this page you can search for capabilities of the role you are currently editing by clicking the *SEARCH* button. The results will be shown below that button. The search can be narrowed down by selecting a specific app in the *App* box, and a namespace of the selected app in the *Namespace* box.

Note: The namespaces for the *Namespace* box can be managed as described in *Namespaces* (page 32).

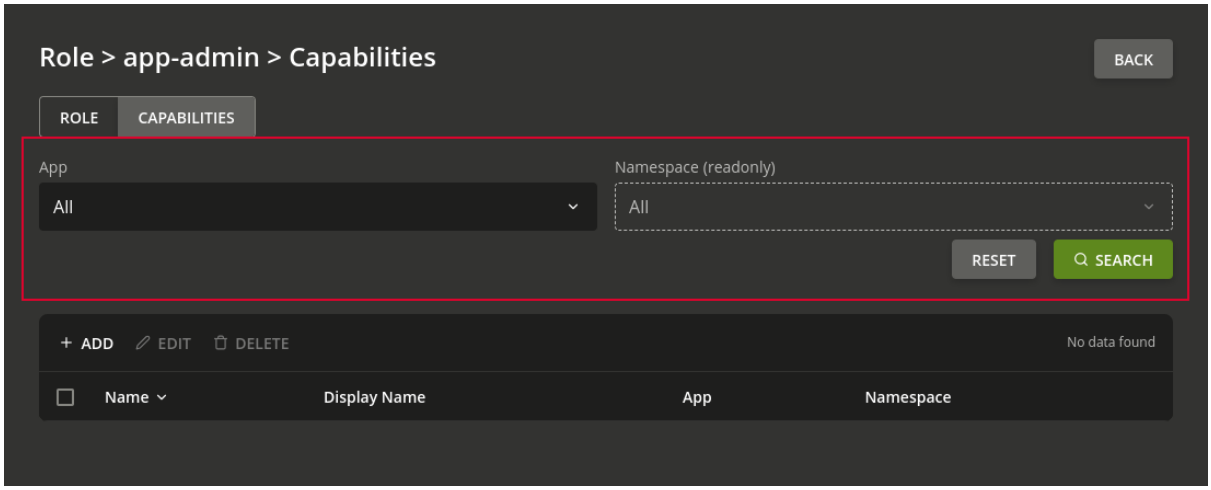


Fig. 5.13: Form elements for the search of capabilities.

5.3.3 Edit a capability of a role

To edit a capability of a role, follow the steps in *Listing and searching capabilities of a role* (page 28) to list them and then click on the name of the capability you want to edit.

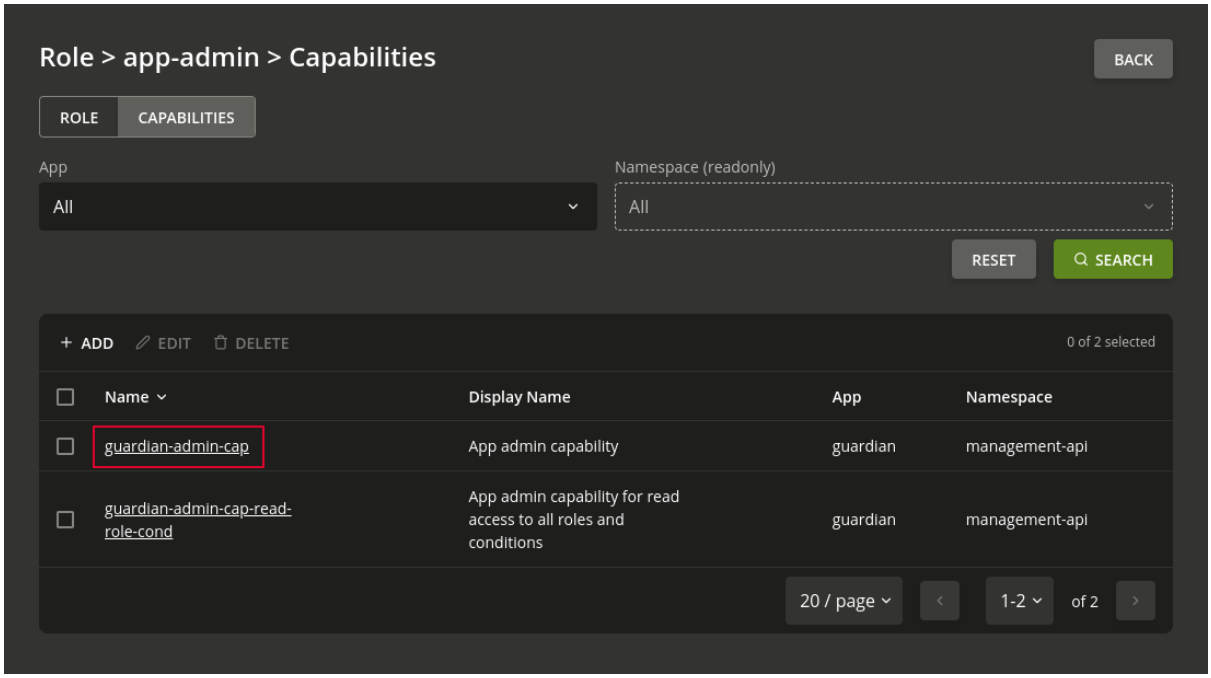


Fig. 5.14: Edit button for listed capabilities.

The page to edit the clicked capability looks like this:

The three noteworthy fields you can edit are the list of *Conditions*, the *Relation* and the list of *Permissions*.

Permissions
: In the *Permissions* list you can edit all permissions the capability will grant if the conditions in the *Conditions* list are met.

Conditions
: In the *Conditions* list you can edit all the conditions that should be checked before the permissions in the *Permissions* list are granted. Some conditions require additional parameters. Additional fields will be shown underneath them once selected.

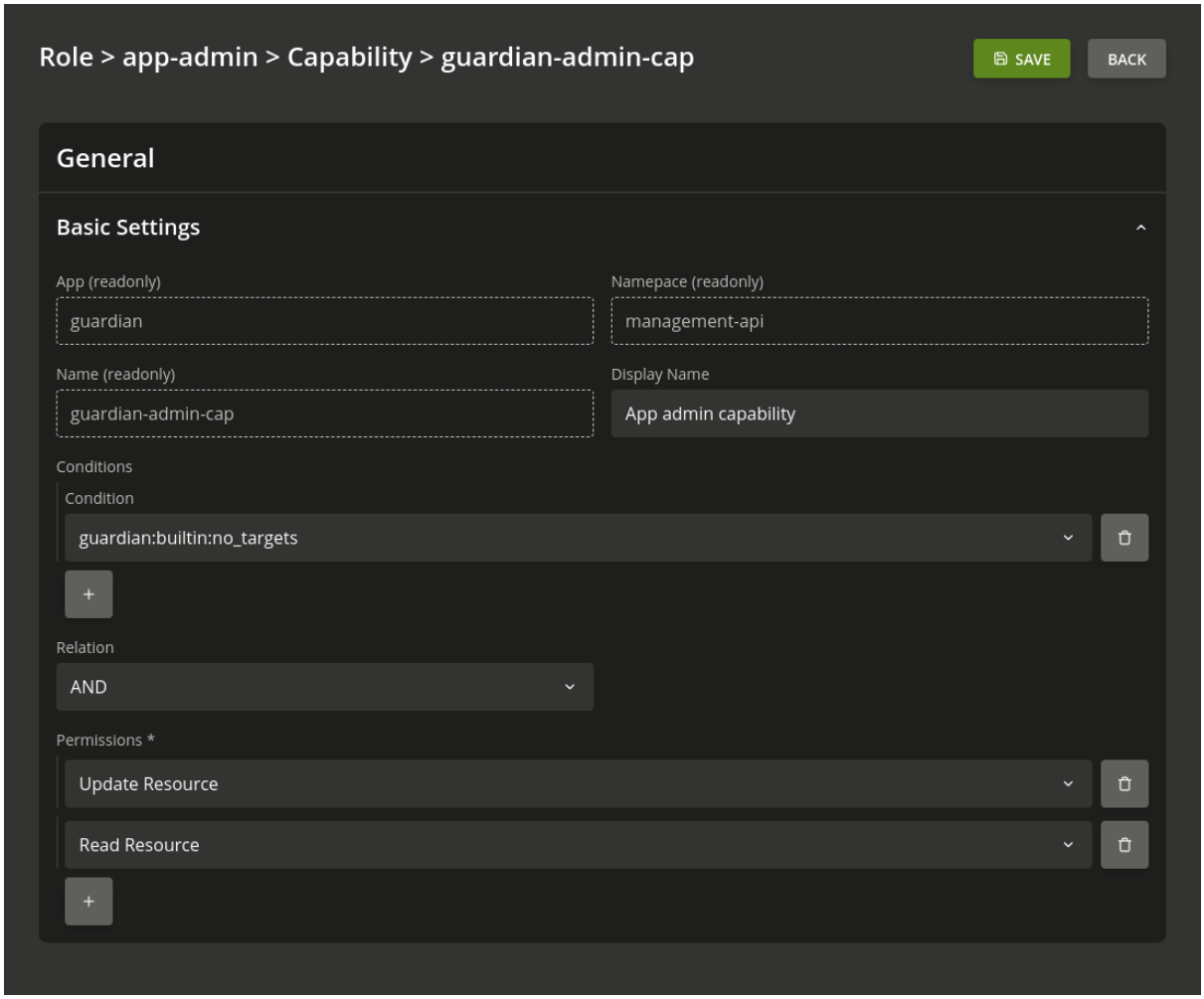


Fig. 5.15: View and edit page of an existing capability.

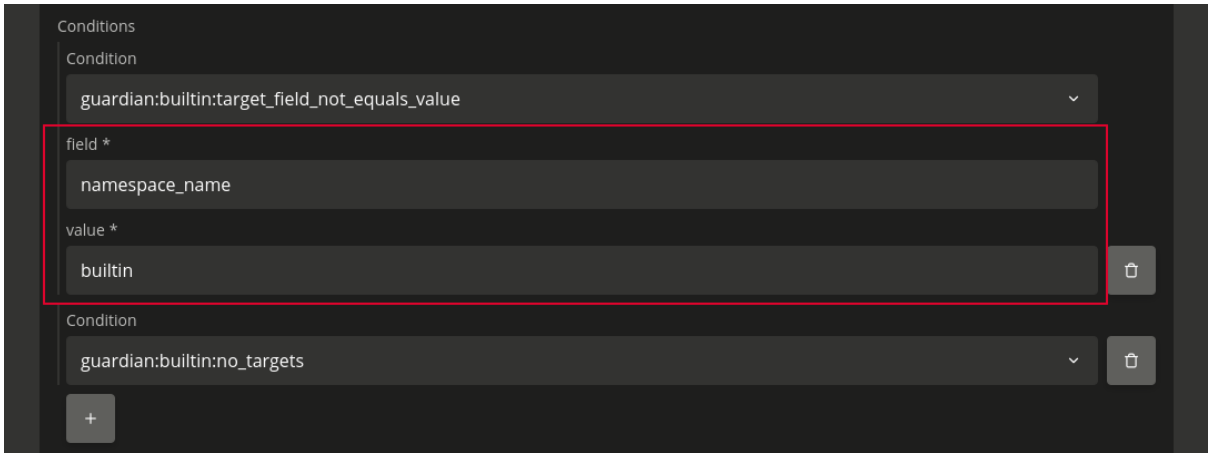


Fig. 5.16: Condition with extra parameters.

Note: See *Conditions Reference* (page 61) for an explanation of the pre-existing conditions.

Relation

The value of the *Relation* box describes in which manner the selected conditions of the *Conditions* should be checked. *AND* means all conditions have to be met, *OR* means only 1 condition has to be met.

5.3.4 Delete capabilities of a role

To delete capabilities, first click on *CAPABILITES* in the navigation menu while editing a role. See *Editing existing roles* (page 25) for more details on editing a role.

Search and select all the capabilities you want to delete, then click the *DELETE* button.

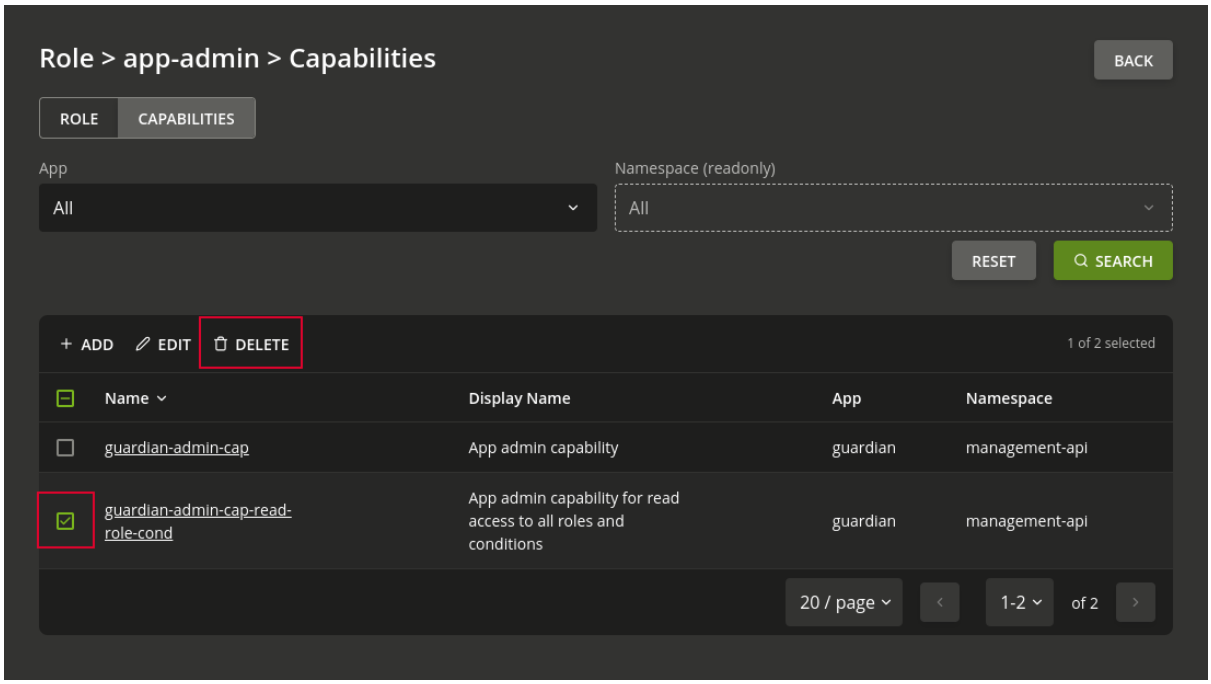


Fig. 5.17: Deletion of capabilities.

5.4 Namespaces

A namespace is a means to categorize roles and permissions. With the **`Guardian Management UI`** namespaces can be created, edited, searched and viewed. For more information about namespaces refer to the *section about Guardian terminology* (page 3).

5.4.1 Create a new namespace

To create a new namespace first open the **`Guardian Management UI`** and click on *NAMESPACES* in the navigation menu.

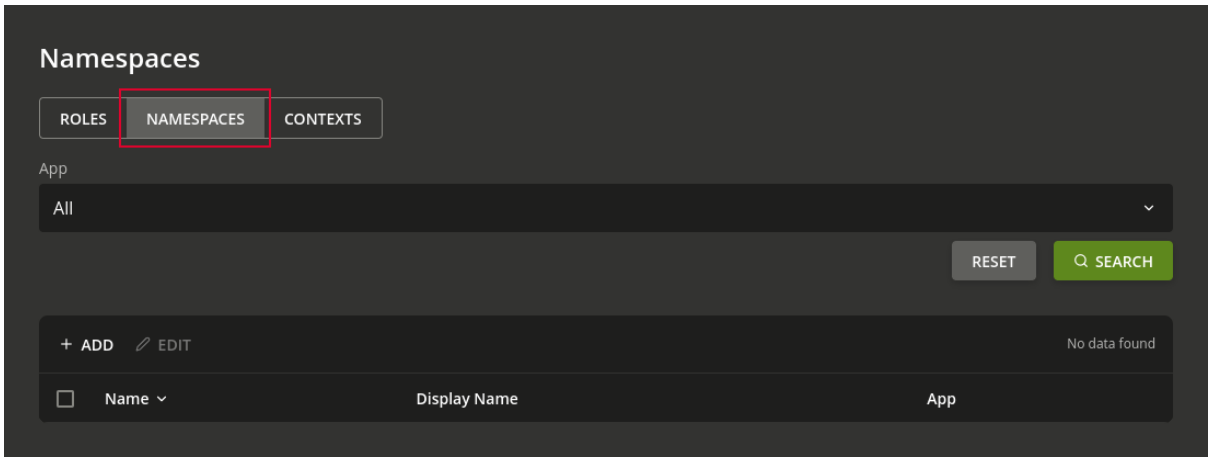


Fig. 5.18: Link to the "Namespaces" page.

Then click on the + *ADD* button to open the page to create a new namespace.

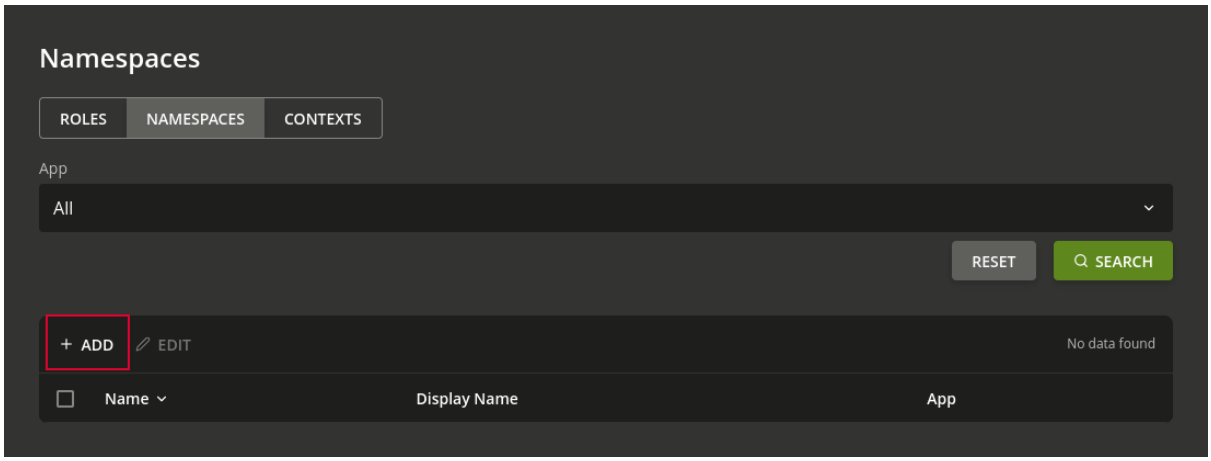


Fig. 5.19: Click + *ADD* to create a new namespace.

The page to create a new namespace looks like this:

Fill out all the necessary fields and click on the *CREATE NAMESPACE* button to create the namespace. A pop-up will be shown which confirms the creation by displaying the namespace name.

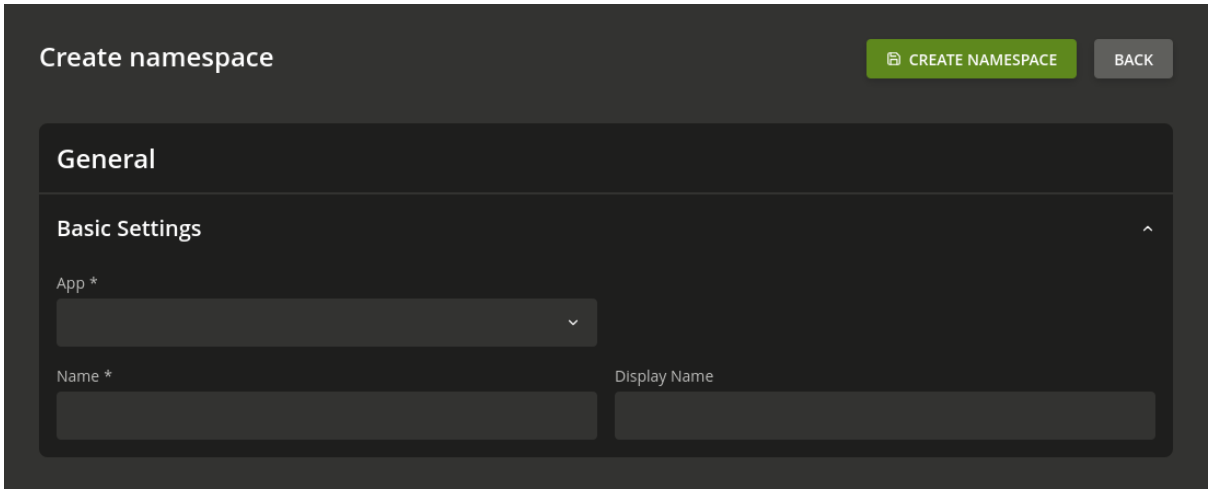


Fig. 5.20: Page to create a new namespace.

5.4.2 Listing and searching namespaces

To list existing namespaces open the **`Guardian Management UI`** and click on *NAMESPACES* in the navigation menu.

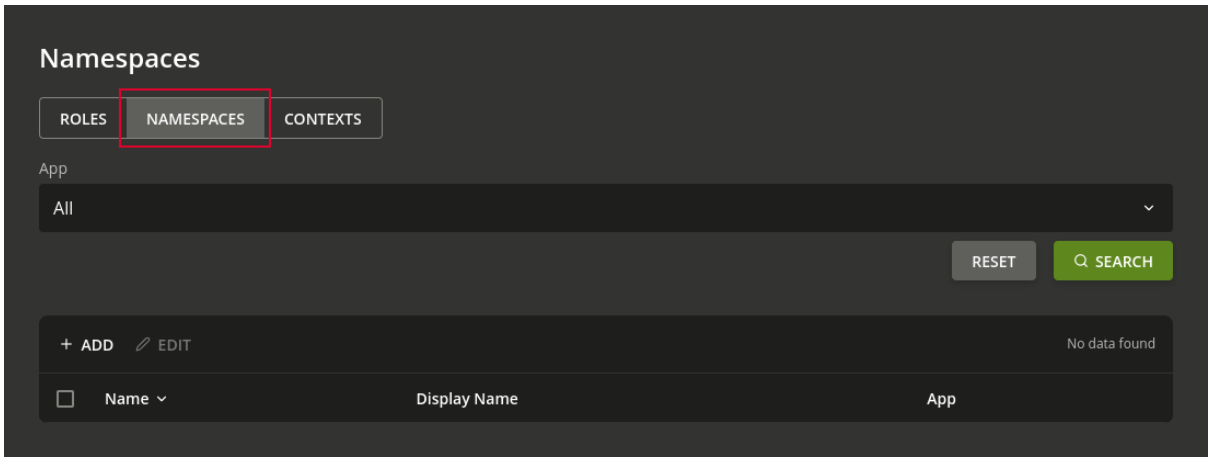


Fig. 5.21: Link to the "Namespaces" page.

On this page you can search for existing namespaces by clicking the *SEARCH* button. The results will be shown below that button. The search can be narrowed down by selecting a specific app in the *App* box.

5.4.3 Editing existing namespaces

To edit a namespaces, follow the steps in *Listing and searching namespaces* (page 33) to list them and then click on the name of the namespace you want to edit.

The page to edit the namespace you clicked looks like this:

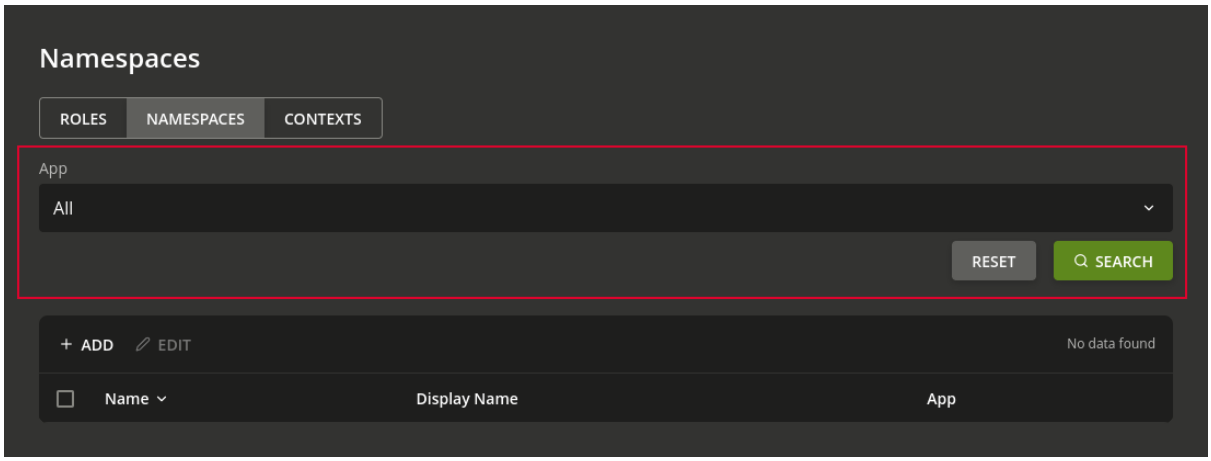


Fig. 5.22: Form elements for the search of namespaces.

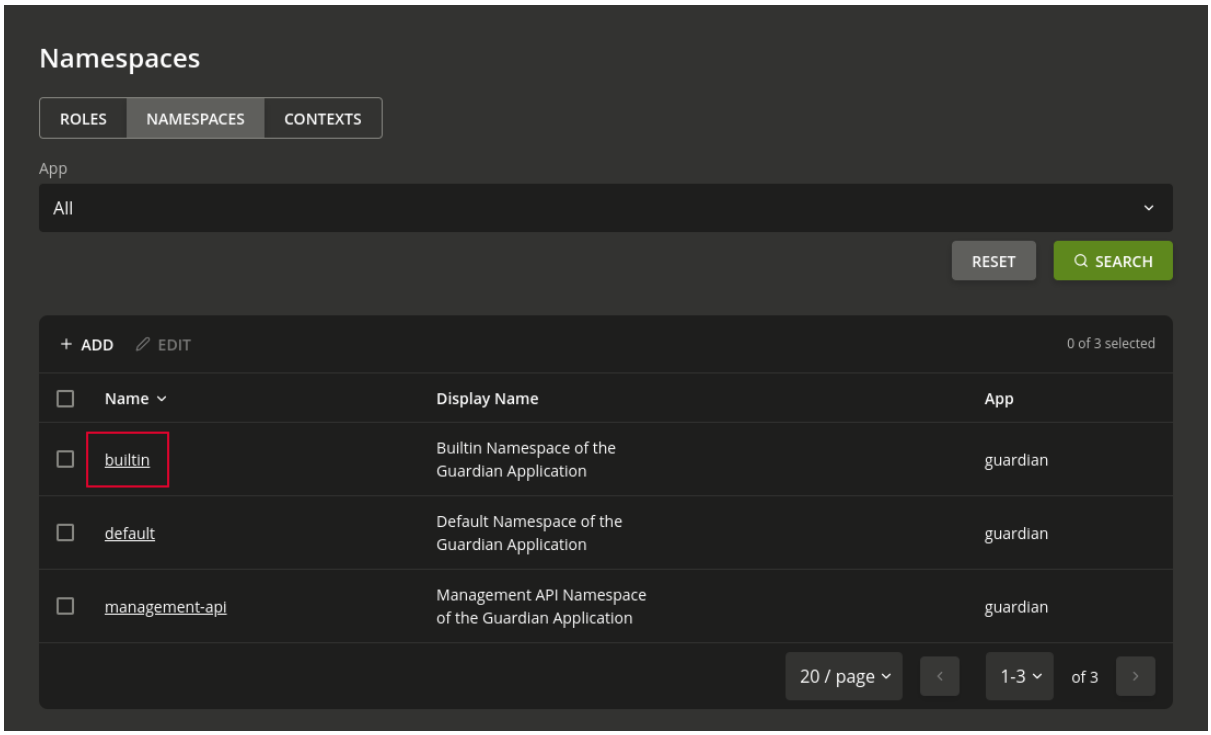


Fig. 5.23: Edit button for listed namespaces.

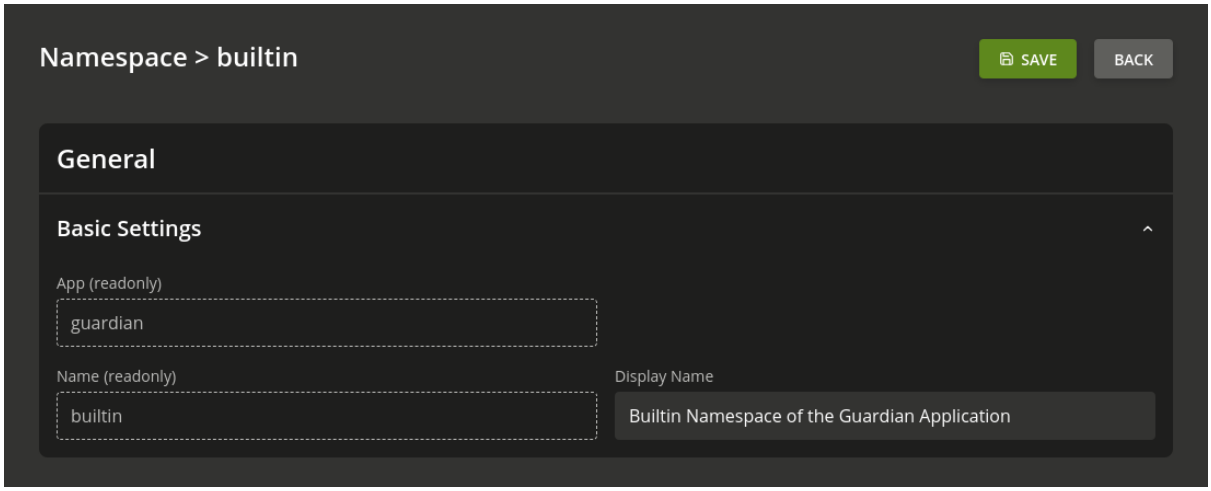


Fig. 5.24: View and edit page of an existing namespace.

5.4.4 Deleting namespaces

Deleting namespaces is not possible at the moment. Neither through the web-interface nor the REST API.

5.5 Contexts

A context is an additional tag that can be applied to a *role*, to make it only apply in certain circumstances. With the **`Guardian Management UI`** you can create, edit, search and view a context. For more information about contexts refer to the *section about Guardian terminology* (page 3).

5.5.1 Create a new context

To create a new context first open the **`Guardian Management UI`** and click on *CONTEXTS* in the navigation menu.

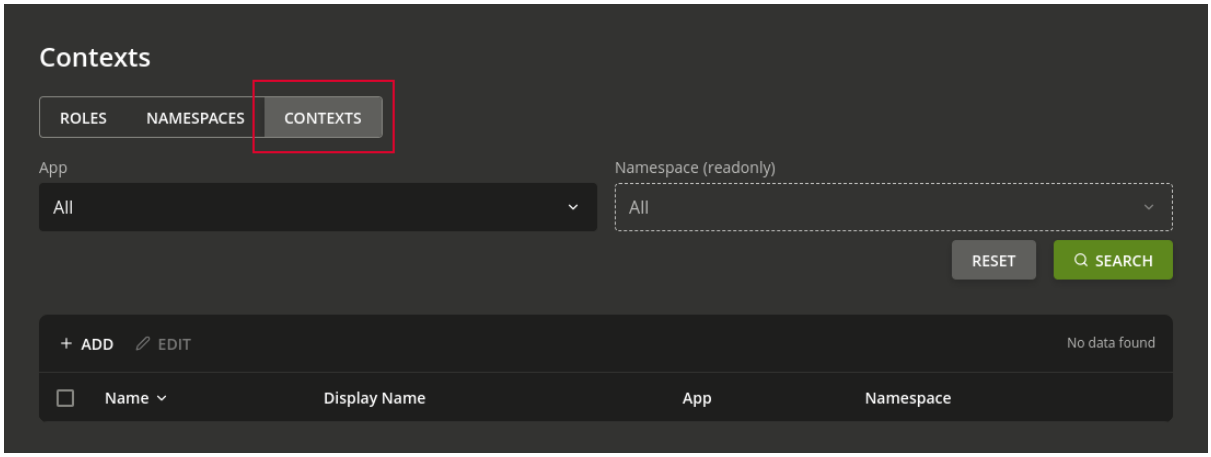


Fig. 5.25: Link to the “Namespaces” page.

Then click on the *ADD* button to open the page to create a new context.

The page to create a new context looks like this:

Fill out all the necessary fields and click on the *CREATE CONTEXT* button to create the context. A pop-up will be shown which confirms the creation by displaying the context name.

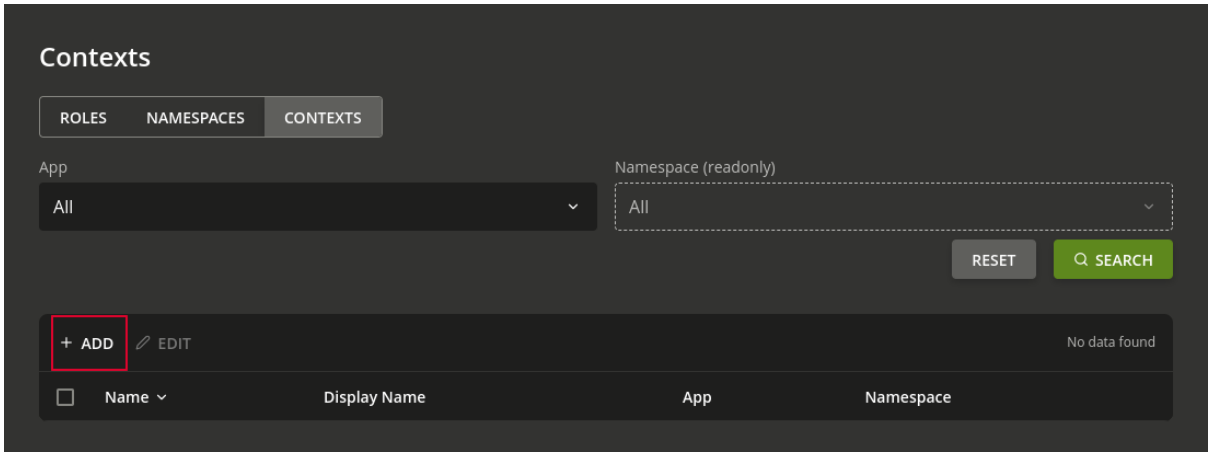


Fig. 5.26: Click + *ADD* to create a new context.

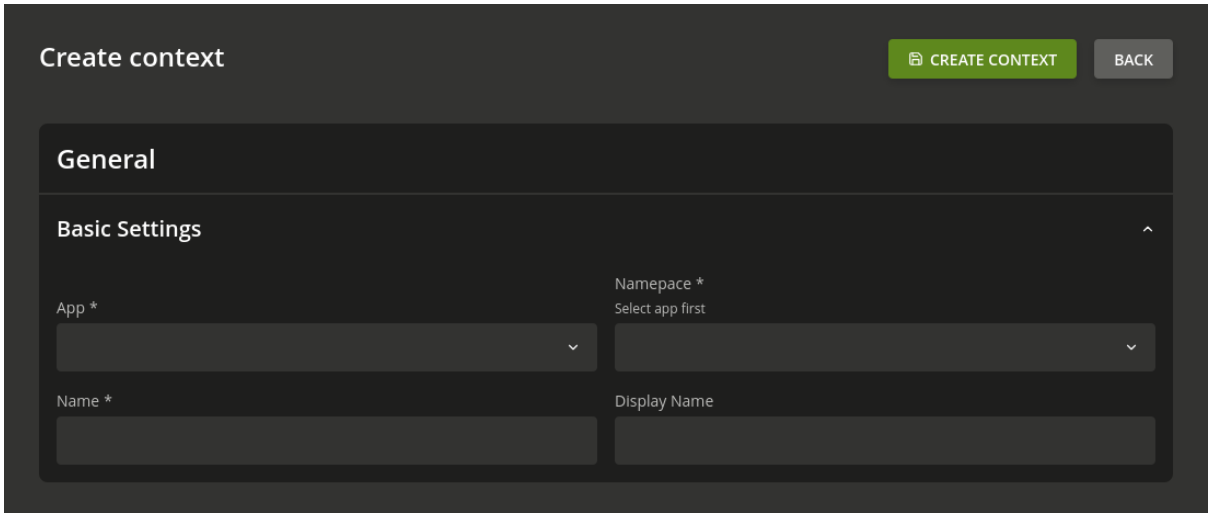


Fig. 5.27: Page to create a new context.

5.5.2 Listing and searching contexts

To list existing contexts open the **`Guardian Management UI`** and click on *CONTEXTS* in the navigation menu.

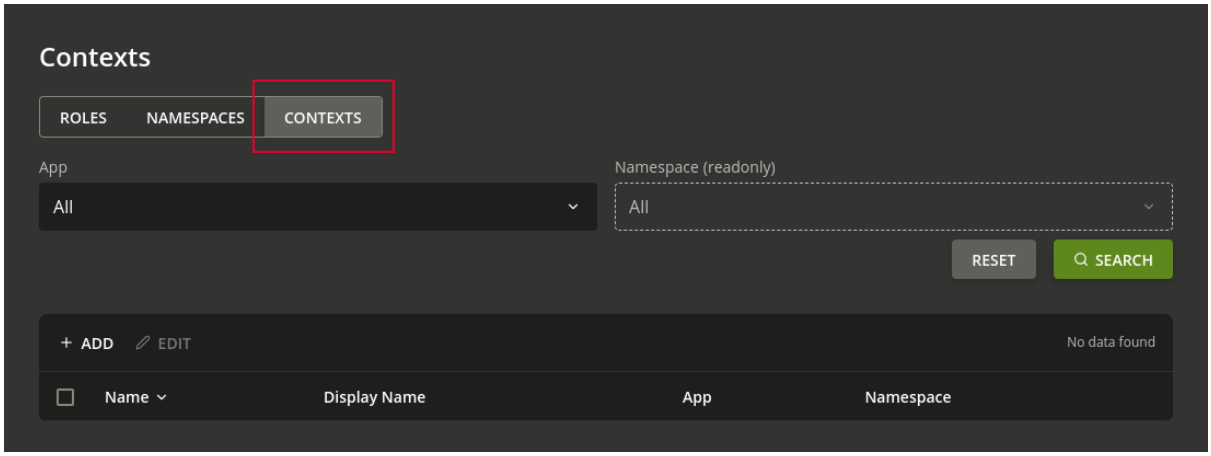


Fig. 5.28: Link to the "Contexts" page.

On this page you can search for existing contexts by clicking the *SEARCH* button. The results will be shown below that button. The search can be narrowed down by selecting a specific app in the *App* box, and a namespace of the selected app in the *Namespace* box.

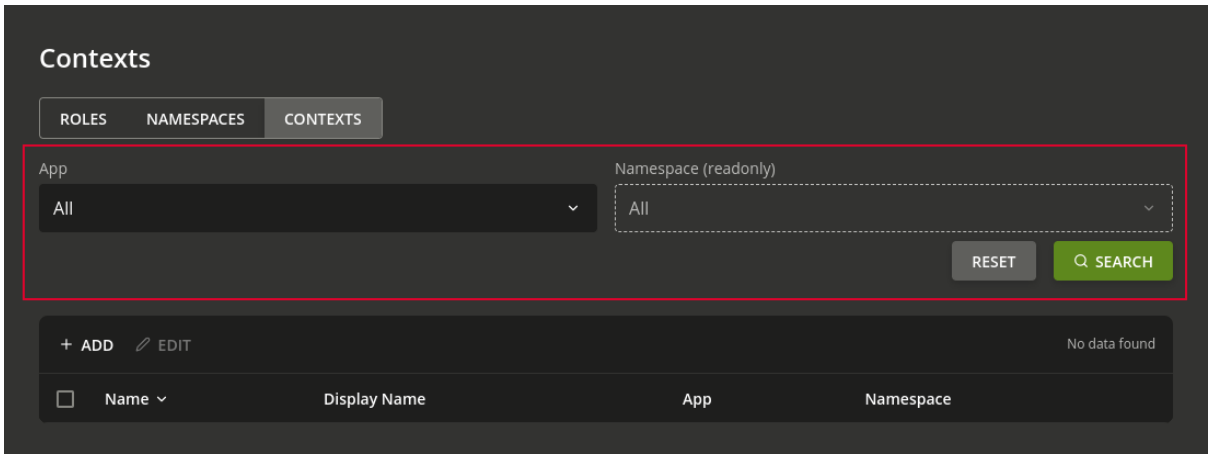


Fig. 5.29: Form elements for the search of contexts.

Note: The namespaces for the *Namespace* box can be managed as described in *Namespaces* (page 32).

5.5.3 Editing existing contexts

To edit a context, follow the steps in *Listing and searching contexts* (page 37) to list them and then click on the name of the context you want to edit.

The page to edit the context you clicked looks like this:

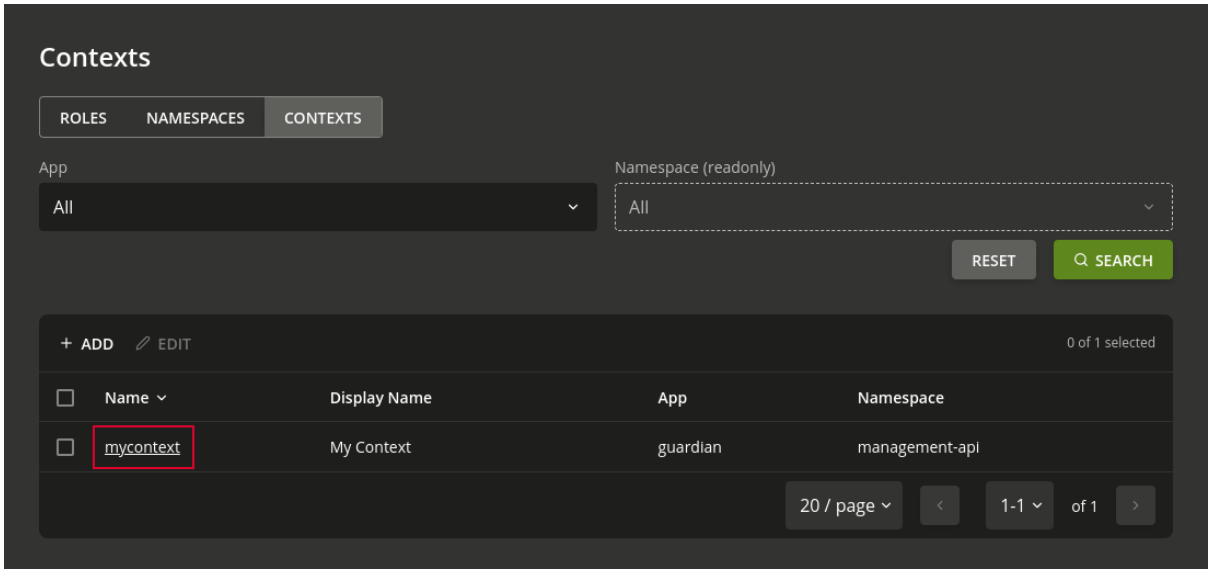


Fig. 5.30: Edit button for listed contexts.

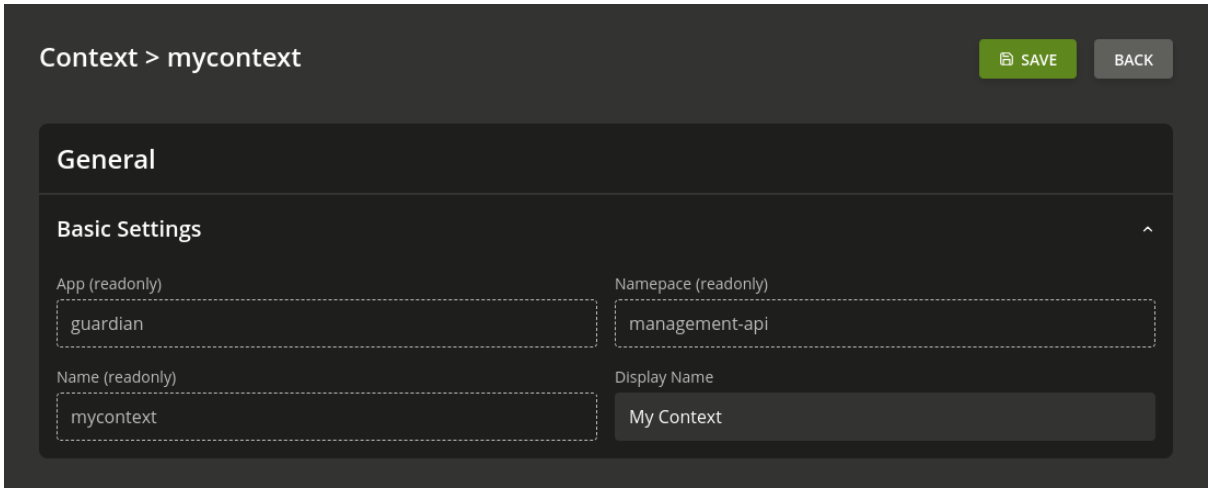


Fig. 5.31: View and edit page of an existing context.

5.5.4 Deleting contexts

Deleting contexts is not possible at the moment. Neither through the web-interface nor the REST API.

CHAPTER

SIX

MANAGEMENT API AND AUTHORIZATION API

Note: This is a highly technical topic, and is primarily geared towards *app developers* who want to integrate an *app* with the Guardian. Familiarity with using the command line and working with an HTTP API, is necessary to understand this chapter.

6.1 Introduction

The *Management API* and *Authorization API* are the two REST[9] APIs[10] for the Guardian.

Please read the *Developer quick start* (page 45) for concrete examples of using the APIs.

6.2 Management API

The *Management API* is a general-purpose CRUD[11] interface for managing Guardian objects. When installing a new *app* that integrates with the Guardian, the join script[12] must register the app and create any new Guardian elements that it needs, using this API.

Once the join script is complete, the app has no more need to contact the Management API. However, *guardian admins* and *guardian app admins* may use this API to modify *roles* and *capabilities* after installing the app.

6.2.1 API documentation

Swagger documentation for the API is located at `/guardian/management/docs` on the server where the *Management API* is installed.

The API requires authentication. Click the *Authorize* button at the top of the page. The default client does not require a `client_secret`. When logging in, please use the credentials of either a *guardian admin* or a *guardian app admin*.

Note: Only the *capabilities* have a `DELETE` endpoint. Please see the chapter on *Limitations* (page 59) for more information.

[9] https://en.wikipedia.org/wiki/REST
[10] https://en.wikipedia.org/wiki/API
[11] https://en.wikipedia.org/wiki/Create,_read,_update_and_delete
[12] https://docs.software-univention.de/developer-reference/latest/en/join/write-join.html#join-write

6.2.2 Guardian naming conventions

When creating a new object in the *Management API*, the `name` for the object should always be lower-case ASCII alphanumeric, with hyphens or underscores to separate words.

For example, if you want to create a *role* for users who manage a pet store, you might name the role `pet-store-manager`.

With the exception of apps and *namespaces* themselves, all objects belong to a namespace. We often represent the full name of an object as a three-part string, with each section separated by colons:

```
<app-name>:<namespace-name>:<object-name>
```

For example, if the `pet-store-manager` role mentioned above belongs to the namespace `stores` for the app `inventory-manager`, then the fully namespaced role is `inventiory-manager:stores:pet-store-manager`.

6.2.3 Registering an app

Before an *app* can use the *Management API*, it needs to register itself at the `/guardian/management/apps/register` endpoint.

Registration looks like:

```
MANAGEMENT_SERVER="$(hostname).$(ucr get domainname)/guardian/management"

curl -X POST \
    -H "Content-Type: application/json" \
    -H "Authorization: Bearer $keycloak_token" \
    -d '{"name":"my-app", "display_name":"My App"}' \
    $MANAGEMENT_SERVER/apps/register
```

Note: There is another endpoint, `/guardian/management/apps` which will also create a new app. However, the `register` endpoint also does additional setup for the app, such as creating a *guardian app admin role* that can be used to manage the app.

Unless you know what you are doing, please avoid the `/guardian/management/apps` endpoint.

After registration, an app must at the bare minimum register the *permissions* that it needs. However, other Guardian objects are optional and may be manually created by a *guardian app admin* later.

6.2.4 Conditions

When constructing a *capability*, the list of available *conditions* is available with a `GET` to the `/guardian/management/conditions` endpoint. Each condition provides a `documentation` string and a list of `parameters` it needs.

Please read the chapter on *Conditions Reference* (page 61) for more information on Guardian's built-in conditions.

If the Guardian does not provide a condition that you need, you can create it through the `/guardian/management/conditions/{app-name}/{namespace-name}` endpoint. This requires a knowledge of Rego[13], and the code must be `base64` encoded when submitting it to the Guardian.

Please see *Registering custom conditions* (page 51) in the *Developer quick start* (page 45) guide.

[13] https://www.openpolicyagent.org/docs/latest/policy-language/

6.2.5 Contexts

Contexts are a special feature of the Guardian that allows *guardian admins* to tell *apps* about where a *role* applies.

For example, if Happy Employees installs the Cake Express app, Happy Employees can create a `london` context and a `berlin` context, which it includes with the `cake-express:cakes:cake-orderer` role. Happy Employees can then create a *capability* where users can only order cakes for people in the same context.

Some of the built-in Guardian *conditions* explicitly support contexts, such as:

- *`target_has_same_context`* (page 62)
- *`target_has_role_in_same_context`* (page 62)
- *`target_does_not_have_role_in_same_context`* (page 61)

An app must explicitly support contexts and send them as part of requests to the *Authorization API*. in order to use contexts within a capability. Apps must specify in their documentation whether or not they support contexts.

6.3 Authorization API

The *Authorization API* helps an *app* determine whether an *actor* is authorized to perform a given action within the app.

6.3.1 API documentation

Swagger documentation for the API is located at `/guardian/authorization/docs` on the server where the *Authorization API* is installed.

The API requires authentication. Click the *Authorize* button at the top of the page. The default client does not require a `client_secret`.

6.3.2 Endpoint overview

There are four primary endpoints in the *Authorization API*:

- `/guardian/authorization/permissions`
- `/guardian/authorization/permissions/with-lookup`
- `/guardian/authorization/permissions/check`
- `/guardian/authorization/permissions/check/with-lookup`

The first two endpoints answer the question "What are all the *permissions* an *actor* has?".

The second two endpoints answer the question "Does the user have a specific set of permissions?". You must supply a list of permissions that you want to check.

In both cases, you must supply an actor, and you may optionally supply *targets* that are used to answer these questions.

About with-lookup endpoints

Some *apps* maintain all their own data in regards to *actors* and *targets*. This means that they do not need access to UDM[14] in order to check *capabilities*. The examples in the *Developer quick start* (page 45) all use endpoints without lookup.

However, endpoints ending in `with-lookup` will search for the actor and targets in UDM and use the results in checking capabilities. To use the UDM lookup feature, supply the LDAP `dn` as the `id` of the actor and targets.

You do not need to supply any `attributes` or `roles` in the request, if you use the `with-lookup` endpoints.

General permissions versus target permissions

The *Authorization API* endpoints allow an *app* to evaluate *permissions* for an *actor*.

A general permission is a permission that exists, regardless of whether there are any `targets` present in the API request. When listing all permissions, you must set `include_general_permissions` to `true` in the request, if you want to see these permissions. See the section on *Listing all general permissions* (page 53) in the *Developer quick start* (page 45) guide for an example.

Target permissions require one or more *targets* to be present in the `targets` field of the request. See the section on *Listing all target permissions* (page 54) in the *Developer quick start* (page 45) guide for an example.

Old target versus new target

When sending `targets` to the *Authorization API*, a *target* consists of an `old_target` and a `new_target`. The `old_target` represents the existing state of the target, and the `new_target` represents the future state of the target.

For example, a *condition* could check that the `new_target` user password is not the same as the `old_target` password.

If the *app* doesn't care about an old and new state of the target, then only the `old_target` is required.

All *built-in conditions* (page 61) check the `old_target`.

6.3.3 Custom endpoints

The *Authorization API* has an experimental endpoint, `/guardian/authorization/{app-name}/{namespace-name}/{endpoint-name}`, that allows an *app* to define its own custom Rego[15] code to evaluate permissions.

The endpoint does not have UDM access, so the app must supply all of its own data for *actors* and *targets*.

This endpoint is not implemented yet, so please do not use it.

[14] https://docs.software-univention.de/developer-reference/latest/en/udm/index.html
[15] https://www.openpolicyagent.org/docs/latest/policy-language/

CHAPTER

SEVEN

DEVELOPER QUICK START

Note: This is a highly technical topic, and is primarily geared towards *app developers* who want to integrate an *app* with the Guardian. Familiarity with using the command line, working with an API, and writing code is necessary to understand this chapter.

You should also be familiar with:

- Documentation for App Center Providers[16]
- Manual for Developers[17]

This section provides a walk-through of the steps necessary to integrate an *app* with the Guardian.

ACME Corporation is an *app developer* who creates Cake Express, which allows people to order cakes for company events, and which can be installed from the Univention App Center. They want to integrate Cake Express with the Guardian.

Note: The example scripts assume that:

- The app is installed on the same server as the Management API, and
- The app is installed on the same server as the Keycloak server.

If either of these two things is not true, you will need to find a way for the UCS *app infrastructure maintainer* to communicate their locations to the script at run time.

7.1 Management API

7.1.1 Getting a Keycloak token

The first thing that ACME Corporation needs to do is to write a join script for their *app*. This app will need to interact with the *Management API*, and to do this the join script must get a token from Keycloak[18] to authenticate all calls to the API.

Here are the variables you need to get a token:

```
binduser=Administrator
bindpwd=password

CLIENT_ID=guardian-scripts

GUARDIAN_KEYCLOAK_URL=$(ucr get guardian-management-api/oauth/keycloak-uri)
```

(continues on next page)

[16] https://docs.software-univention.de/app-center/latest/en/contents.html
[17] https://docs.software-univention.de/developer-reference/latest/en/contents.html
[18] https://docs.software-univention.de/keycloak-app/latest/#doc-entry

(continued from previous page)

```
SYSTEM_KEYCLOAK_URL=$(ucr get keycloak/server/sso/fqdn)
KEYCLOAK_BASE_URL=${GUARDIAN_KEYCLOAK_URL:-$SYSTEM_KEYCLOAK_URL}

KEYCLOAK_URL="$KEYCLOAK_BASE_URL/realms/ucs/protocol/openid-connect/token"
if [[ ! $KEYCLOAK_URL == http ]]; then
    KEYCLOAK_URL="https://$KEYCLOAK_URL"
fi
```

Note: In a typical join script, the `--binduser`, `--bindpwd`, and `--bindpwdfile` are available, which specify an administrator user, and either a password or a file for parsing the password.

The example above assumes that the join script has already parsed these parameters into `binduser` and `bindpwd` variables.

You can retrieve the token with:

```
token=$(curl -d "client_id=$CLIENT_ID" \
     -d "username=$binduser" \
     -d "password=$bindpwd" \
     -d "grant_type=password" \
     $KEYCLOAK_URL | sed 's/.*"access_token":"\([[:alnum:]\.-_-]*\)".*/\1/')
```

The `token` is referenced in all commands for subsequent sections. You may need to refresh the token several times, if you are entering commands manually.

7.1.2 Registering an app

ACME Corporation now needs to let the Guardian know about its *app*, Cake Express. To do this, it needs to take the token from the *previous section* (page 45) and make a request to the *Management API*.

```
MANAGEMENT_SERVER="$(hostname).$(ucr get domainname)/guardian/management"

curl -X POST \
    -H "Content-Type: application/json" \
    -H "Authorization: Bearer $token" \
    -d '{"name":"cake-express", "display_name":"Cake Express"}' \
    $MANAGEMENT_SERVER/apps/register
```

Note: All names in Guardian are lower-case ASCII alphanumeric with either underscores or hyphens. The encoding for display names is only limited by the character support for the PostgreSQL database that Guardian uses.

ACME Corporation is now ready to start setting up the Guardian to work with Cake Express.

7.1.3 Registering namespaces

A *namespace* is just a handy categorization to store everything that an *app* wants to use in Guardian, like *roles* and *permissions*.

Every app gets a `default` namespace to use. But ACME Corporation wants to manage three different facets of Cake Express:

- `cakes`: Category for everything related to what is actually being sold.
- `orders`: Category for administration of orders.
- `users`: Category for managing other users of Cake Express.

Later, ACME Corporation will create some roles in each of these namespaces for doing tasks in Cake Express.

Here is how ACME Corporation creates these namespaces:

```
curl -X POST \
    -H "Content-Type: application/json" \
    -H "Authorization: Bearer $token" \
    -d '{"name":"cakes", "display_name":"Cakes"}' \
    $MANAGEMENT_SERVER/namespaces/cake-express
```

```
curl -X POST \
    -H "Content-Type: application/json" \
    -H "Authorization: Bearer $token" \
    -d '{"name":"orders", "display_name":"Orders"}' \
    $MANAGEMENT_SERVER/namespaces/cake-express
```

```
curl -X POST \
    -H "Content-Type: application/json" \
    -H "Authorization: Bearer $token" \
    -d '{"name":"users", "display_name":"Users"}' \
    $MANAGEMENT_SERVER/namespaces/cake-express
```

7.1.4 Registering roles

ACME Corporation wants to create three different *roles* for users of Cake Express:

- `cake-express:cakes:cake-orderer`: Someone who can order cakes from Cake Express.
- `cake-express:orders:finance-manager`: Someone who manages the expenses for the orders.
- `cake-express:users:user-manager`: Someone who manages other users within Cake Express.

ACME Corporation also wants to create a role for some of their cakes:

- `cake-express:cakes:birthday-cake`: A cake just for employee birthdays.

Each role above consists of the following parts, separated by a `:`:

- *app*: e.g., `cake-express`
- *namespace*: e.g., `cakes`
- role name: e.g., `cake-orderer`

Here is how ACME Corporation creates these roles:

```
curl -X POST \
    -H "Content-Type: application/json" \
    -H "Authorization: Bearer $token" \
    -d '{"name":"cake-orderer", "display_name":"Cake Orderer"}' \
    $MANAGEMENT_SERVER/roles/cake-express/cakes
```

```
curl -X POST \
    -H "Content-Type: application/json" \
    -H "Authorization: Bearer $token" \
    -d '{"name":"finance-manager", "display_name":"Finance Manager"}' \
    $MANAGEMENT_SERVER/roles/cake-express/orders
```

```
curl -X POST \
    -H "Content-Type: application/json" \
    -H "Authorization: Bearer $token" \
    -d '{"name":"user-manager", "display_name":"User Manager"}' \
    $MANAGEMENT_SERVER/roles/cake-express/users
```

```
curl -X POST \
    -H "Content-Type: application/json" \
    -H "Authorization: Bearer $token" \
    -d '{"name":"birthday-cake", "display_name":"Birthday Cake"}' \
    $MANAGEMENT_SERVER/roles/cake-express/cakes
```

7.1.5 Registering permissions

ACME Corporation wants to provide some *permissions* that define what users of Cake Express want to do:

- `cake-express:cakes:order-cake`: Users with this permission are allowed to order cakes.
- `cake-express:orders:cancel-order`: Users can cancel a cake order.
- `cake-express:users:manage-notifications`: Users can manage cake notifications.

Here is how ACME Corporation creates these permissions:

```
curl -X POST \
    -H "Content-Type: application/json" \
    -H "Authorization: Bearer $token" \
    -d '{"name":"order-cake", "display_name":"order cake"}' \
    $MANAGEMENT_SERVER/permissions/cake-express/cakes
```

```
curl -X POST \
    -H "Content-Type: application/json" \
    -H "Authorization: Bearer $token" \
    -d '{"name":"cancel-order", "display_name":"cancel order"}' \
    $MANAGEMENT_SERVER/permissions/cake-express/orders
```

```
curl -X POST \
    -H "Content-Type: application/json" \
    -H "Authorization: Bearer $token" \
    -d '{"name":"manage-notifications", "display_name":"manage notifications"}' \
    $MANAGEMENT_SERVER/permissions/cake-express/users
```

7.1.6 Registering capabilities

Finally, ACME Corporation wants to define some default *capabilities* for their applications. The *guardian app admin* that installs Cake Express can change these later, but these default capabilities make it easier for Cake Express to work out of the box.

They want to create:

1. Users with the `cake-orderer` role are allowed to order cakes.
2. Users with the `finance-manager` role, or the person who ordered the cake, have the permission to cancel the cake order.
3. Users with the `user-manager` role have the permission to manage cake notifications. Users can also manage their own notifications for cakes that are sent to them, except for notifications related to birthday cakes.

Here is how ACME Corporation creates the capability for ordering cake:

```
curl -X POST \
    -H "Content-Type: application/json" \
    -H "Authorization: Bearer $token" \
    -d '{
          "name": "cake-orderer-can-order-cake",
          "display_name": "Cake Orderers can order cake",
```

(continues on next page)

(continued from previous page)

```
          "role": {
            "app_name": "cake-express",
            "namespace_name": "cakes",
            "name": "cake-orderer"
          },
          "conditions": [],
          "relation": "AND",
          "permissions": [
            {
              "app_name": "cake-express",
              "namespace_name": "cakes",
              "name": "order-cake"
            }
          ]
        }' \
    $MANAGEMENT_SERVER/capabilities/cake-express/cakes
```

Here is how ACME Corporation creates the capability for canceling an order. This requires two `POST` requests in order to create it:

```
curl -X POST \
    -H "Content-Type: application/json" \
    -H "Authorization: Bearer $token" \
    -d '{
          "name": "finance-manager-can-cancel-order",
          "display_name": "Finance Manager can cancel orders",
          "role": {
            "app_name": "cake-express",
            "namespace_name": "orders",
            "name": "finance-manager"
          },
          "conditions": [],
          "relation": "AND",
          "permissions": [
            {
              "app_name": "cake-express",
              "namespace_name": "orders",
              "name": "cancel-order"
            }
          ]
        }' \
    $MANAGEMENT_SERVER/capabilities/cake-express/orders
```

```
curl -X POST \
    -H "Content-Type: application/json" \
    -H "Authorization: Bearer $token" \
    -d '{
          "name": "self-can-cancel-order",
          "display_name": "Users can cancel their own order",
          "role": {
            "app_name": "cake-express",
            "namespace_name": "cakes",
            "name": "cake-orderer"
          },
          "conditions": [
            {
              "app_name": "guardian",
              "namespace_name": "builtin",
              "name": "target_field_equals_actor_field",
              "parameters": [
                {
```

(continues on next page)

(continued from previous page)

```
                "name": "actor_field",
                "value": "id"
              },
              {
                "name": "target_field",
                "value": "orderer_id"
              }
            ]
          }
        ],
        "relation": "AND",
        "permissions": [
          {
            "app_name": "cake-express",
            "namespace_name": "orders",
            "name": "cancel-order"
          }
        ]
      }' \
    $MANAGEMENT_SERVER/capabilities/cake-express/orders
```

Here is how ACME Corporation creates the capability for managing notifications. This also requires two `POST` requests in order to create it:

```
curl -X POST \
    -H "Content-Type: application/json" \
    -H "Authorization: Bearer $token" \
    -d '{
        "name": "user-manager-can-manage-notifications",
        "display_name": "User Managers can manage cake notifications",
        "role": {
          "app_name": "cake-express",
          "namespace_name": "users",
          "name": "user-manager"
        },
        "conditions": [],
        "relation": "AND",
        "permissions": [
          {
            "app_name": "cake-express",
            "namespace_name": "users",
            "name": "manage-notifications"
          }
        ]
      }' \
    $MANAGEMENT_SERVER/capabilities/cake-express/users
```

```
curl -X POST \
    -H "Content-Type: application/json" \
    -H "Authorization: Bearer $token" \
    -d '{
        "name": "self-can-manage-notifications",
        "display_name": "Users can manage their own notifications, except for␣
↪birthday cakes",
        "role": {
          "app_name": "cake-express",
          "namespace_name": "cakes",
          "name": "cake-orderer"
        },
        "conditions": [
          {
```

(continues on next page)

(continued from previous page)

```
            "app_name": "guardian",
            "namespace_name": "builtin",
            "name": "target_field_equals_actor_field",
            "parameters": [
              {
                "name": "actor_field",
                "value": "id"
              },
              {
                "name": "target_field",
                "value": "recipient_id"
              }
            ]
          },
          {
            "app_name": "guardian",
            "namespace_name": "builtin",
            "name": "target_does_not_have_role",
            "parameters": [
              {
                "name": "role",
                "value": "cake-express:cakes:birthday-cake"
              }
            ]
          }
        ],
        "relation": "AND",
        "permissions": [
          {
            "app_name": "cake-express",
            "namespace_name": "users",
            "name": "manage-notifications"
          }
        ]
      }' \
  $MANAGEMENT_SERVER/capabilities/cake-express/users
```

ACME Corporation is now done with the join script and is ready to start using Guardian with their application.

7.1.7 Registering custom conditions

The Guardian comes with several built-in *conditions*, which are documented in the chapter on *Conditions Reference* (page 61).

However, some *apps* need to write their own custom conditions, and the *Management API* provides an endpoint to facilitate this. The endpoint requires knowledge of Rego[19].

Suppose that ACME Corporation tracks whether or not a user likes cake, and wants to provide a simple condition to *guardian app admins* that allows them to opt users out of receiving a cake, without having to know how Cake Express stores its cake preferences.

The Rego code for this condition is as follows:

```
package guardian.conditions

import future.keywords.if
import future.keywords.in

condition("cake-express:users:recipient-likes-cakes", _, condition_data) if {
```

(continues on next page)

[19] https://www.openpolicyagent.org/docs/latest/policy-language/

(continued from previous page)

```
    condition_data.target.old.attributes.recipient["likes_cakes"]
} else = false
```

You can test this code in the Rego Playground[20] provided by the Open Policy Agent:

```
package guardian.conditions

import future.keywords.if
import future.keywords.in

condition("cake-express:users:recipient-likes-cakes", _, condition_data) if {
    condition_data.target.old.attributes.recipient["likes_cakes"]
} else = false

result := condition("cake-express:users:recipient-likes-cakes", {}, {"target": {
↪"old": {"attributes": {"recipient": {"likes_cakes": true}}}}})
```

Click the *Evaluate* button on the Rego Playground to receive a `true` result.

The code must be `base64` encoded before sending to the API. Here is how ACME Corporation creates a custom condition:

```
curl -X POST \
    -H "Content-Type: application/json" \
    -H "Authorization: Bearer $token" \
    -d '{
          "name": "recipient-likes-cakes",
          "display_name": "recipient likes cakes",
          "documentation": "True if the user recieving a cake likes cakes",
          "parameters": [],
          "code":
↪"cGFja2FnZSBndWFyZGlhbi5jb25kaXRpb25zCgppbXBvcnQgZnV0dXJlLmtleXdvcmRzLmlmCmltcG9ydCBmdXR1cmUua2V
↪"
        }' \
    $MANAGEMENT_SERVER/conditions/cake-express/users
```

ACME Corporation then updates the existing *capability* for ordering cakes:

```
curl -X PUT \
    -H "Content-Type: application/json" \
    -H "Authorization: Bearer $token" \
    -d '{
          "display_name": "Cake Orderers can order cake",
          "role": {
            "app_name": "cake-express",
            "namespace_name": "cakes",
            "name": "cake-orderer"
          },
          "conditions": [
            {
              "app_name": "cake-express",
              "namespace_name": "users",
              "name": "recipient-likes-cakes",
              "parameters": []
            }
          ],
          "relation": "AND",
          "permissions": [
            {
              "app_name": "cake-express",
```

(continues on next page)

[20] https://play.openpolicyagent.org/

(continued from previous page)

```
            "namespace_name": "cakes",
            "name": "order-cake"
          }
        ]
      }' \
  $MANAGEMENT_SERVER/capabilities/cake-express/cakes/cake-orderer-can-order-cake
```

7.2 Authorization API

Please follow the previous section for the *Management API* (page 45) before starting this section.

Note: Code in this section is not part of the join script. This means that it does not have access to the `guardian-scripts` client and Administrator password. As part of the join script for your *app*, you should create your own Keycloak client to use with your app, that allows service accounts and requires a client secret.

All examples in this section use a hypothetical Keycloak client that Cake Express already has.

7.2.1 Listing all general permissions

Cake Express has three tabs in the web interface:

- *Order a Cake*
- *Manage Existing Orders*
- *Settings*

Cake Express uses its own internal rules:

- The *Settings* tab is always available.
- *Order a Cake* is only available to users who are allowed to order cakes and have the `cake-express:cakes:order-cake` permission.
- *Manage Existing Orders* is only available to users who can manage all orders and have the `cake-express:orders:manage-order` permission. Users who can't manage all orders have to use the *Order a Cake* tab to see their own orders.

Alice is a user with id `alice`. She has the `cake-express:cakes:cake-orderer` *role*. Bob has ordered her an anniversary cake, because she has been with the Happy Employees company for 10 years. It is also Alice's birthday in two weeks, so Carol has also ordered her a birthday cake.

Alice logs into Cake Express, and Cake Express needs to know which tabs to show Alice. So Cake Express asks the *Authorization API* for all *capabilities* related to the `cakes` and `orders` namespaces:

```
AUTHORIZATION_SERVER="$(hostname).$(ucr get domainname)/guardian/authorization"

curl -X POST \
    -H "Content-Type: application/json" \
    -H "Authorization: Bearer $token" \
    -d '{
          "namespaces": [
            {
              "app_name": "cake-express",
              "name": "cakes"
            },
            {
              "app_name": "cake-express",
```

(continues on next page)

(continued from previous page)

```
            "name": "orders"
          }
        ],
        "actor": {
          "id": "alice",
          "roles": [
            {
              "app_name": "cake-express",
              "namespace_name": "cakes",
              "name": "cake-orderer"
            }
          ],
          "attributes": {}
        },
        "targets": [],
        "include_general_permissions": true,
        "extra_request_data": {}
      }' \
    $AUTHORIZATION_SERVER/permissions
```

Note: Usually the Authorization API expects one or more targets in order to evaluate permissions. However, you can ask for `general_permissions`, which means the Authorization API will also evaluate all capabilities without a target.

In the Cake Express example of the web interface tabs, we don't have specific objects like cakes to check. We just want to know general permissions, so we set `include_general_permissions` to `true`.

The Authorization API says that Alice has one general permission, `cake-express:cakes:order-cakes`. This means that Cake express should show her the *Order a Cake* tab, but not the *Manage Existing Orders* tab. Cake Express always shows the *Settings* tab.

7.2.2 Listing all target permissions

Now Alice wants to manage her cake notifications, so she clicks on the *Settings* tab and goes to the *Cake Notifications* section.

From the previous call to the API, Cake Express already knows that Alice does not have the `cake-express:users:manage-notifications` general permission for any cake. But Alice might be able to manage notifications for cakes she is associated with. So Cake Express gathers a list of all cakes where Alice is the recipient, and asks the Authorization API for target permissions for those cakes:

```
curl -X POST \
    -H "Content-Type: application/json" \
    -H "Authorization: Bearer $token" \
    -d '{
          "namespaces": [
            {
              "app_name": "cake-express",
              "name": "users"
            }
          ],
          "actor": {
            "id": "alice",
            "roles": [
              {
                "app_name": "cake-express",
                "namespace_name": "cakes",
                "name": "cake-orderer"
```

(continues on next page)

(continued from previous page)

```
            }
          ],
          "attributes": {
            "id": "alice"
          }
        },
        "targets": [
          {
            "old_target": {
              "id": "anniversary-cake-from-bob",
              "roles": [],
              "attributes": {
                "id": "anniversary-cake-from-bob",
                "orderer_id": "bob",
                "recipient_id": "alice",
                "notifications": true
              }
            }
          },
          {
            "old_target": {
              "id": "birthday-cake-from-carol",
              "roles": [
                {
                  "app_name": "cake-express",
                  "namespace_name": "cakes",
                  "name": "birthday-cake"
                }
              ],
              "attributes": {
                "id": "birthday-cake-from-carol",
                "orderer_id": "carol",
                "recipient_id": "alice",
                "notifications": true
              }
            }
          }
        ],
        "include_general_permissions": false,
        "extra_request_data": {}
      }' \
  $AUTHORIZATION_SERVER/permissions
```

Note: *Targets* for the Authorization API can check the `old_target`, which is the original state of the target, and the `new_target`, which is the updated state of the target.

In the case of showing Alice which cakes she can manage, the cakes haven't changed, so the request only needs to supply the `old_target`.

The Authorization API shows that Alice has `cake-express:users:manage-notifications` permissions for the anniversary cake from Bob, but no permissions for the birthday cake from Carol. So Cake Express only shows Alice the anniversary cake from Bob.

7.2.3 Checking specific permissions

When Alice turns notifications off for the anniversary cake, Cake Express makes a confirmation check to make sure she can manage notifications on the cake:

```
curl -X POST \
    -H "Content-Type: application/json" \
    -H "Authorization: Bearer $token" \
    -d '{
          "namespaces": [
            {
              "app_name": "cake-express",
              "name": "users"
            }
          ],
          "actor": {
            "id": "alice",
            "roles": [
              {
                "app_name": "cake-express",
                "namespace_name": "cakes",
                "name": "cake-orderer"
              }
            ],
            "attributes": {
              "id": "alice"
            }
          },
          "targets": [
            {
              "old_target": {
                "id": "anniversary-cake-from-bob",
                "roles": [],
                "attributes": {
                  "id": "anniversary-cake-from-bob",
                  "orderer_id": "bob",
                  "recipient_id": "alice",
                  "notifications": true
                }
              },
              "new_target": {
                "id": "anniversary-cake-from-bob",
                "roles": [],
                "attributes": {
                  "id": "anniversary-cake-from-bob",
                  "orderer_id": "bob",
                  "recipient_id": "alice",
                  "notifications": false
                }
              }
            }
          ],
          "targeted_permissions_to_check": [
              {
                "app_name": "cake-express",
                "namespace_name": "users",
                "name": "manage-notifications"
              }
            ],
            "general_permissions_to_check": [
              {
                "app_name": "cake-express",
```

(continues on next page)

(continued from previous page)

```
                "namespace_name": "users",
                "name": "manage-notifications"
              }
            ],
          "extra_request_data": {}
        }' \
    $AUTHORIZATION_SERVER/permissions/check
```

The Authorization API says that Alice doesn't have general permissions to manage notifications, but she does have permissions for all targets. So Cake Express saves the new notification settings, and Alice will no longer get notifications about her anniversary cake.

CHAPTER

EIGHT

LIMITATIONS

The Guardian software stack is a new product that is developed iteratively. This chapter documents the known limitations of each component.

8.1 Guardian Management API

8.1.1 App Center database limitations

Due to limitations in the Univention App Center, the Guardian Management API should only be deployed once in any UCS domain. This is due to the fact that each instance of the app gets its own database for the persistent data. This would mean that every instance has its own set of *apps*, *conditions*, *roles*, etc. The App Center does not prevent anyone from deploying as many instances of the Guardian Management API as desired, so this limitation has to be kept in mind.

8.1.2 No object deletion

The Management API does not allow for the deletion of objects at the moment, with the exception of *capabilities*. This is due to the relation of the different object types with each other and the complex consistency checks this operation would entail.

8.1.3 Policy endpoint is public

The endpoint in the Management API where the Authorization API can download the policy data for decision making can be accessed without any authentication. Therefore all data that is contained in the Management API has to be considered public information.

8.2 Guardian Authorization API

8.2.1 Limitation for `with-lookup` endpoints

The Guardian generally allows each client *application* to use its own structure for data that is used for authorization. As long as the capabilities and conditions are created in a fashion that handles data correctly, there are no restrictions what the data must look like.

However, the `with-lookup` endpoints, which allow the Authorization API to fetch data from UDM on behalf of the *app*, are limited to the structure of *actors* and *targets* returned by the UDM REST API.

8.3 Guardian Management UI

8.3.1 Frontend-only pagination

The Management UI in its current state always fetches all objects in their respective list views. This might reduce performance in the UI if working with very big datasets.

8.3.2 No typing for condition parameters

When managing the *capabilities* of a *role* in the UI and editing the *conditions*, the parameters of those conditions are currently not typed. Therefore it is important to take special care when entering the values for condition parameters.

If there are any problems with users not having the correct permissions as configured, it should be one of the first places to check. Make sure that there are no errors due to wrongly typed parameter values.

8.3.3 UCS Portal integration

The Management UI can be accessed from the UCS Portal, but is opened in a new tab. Currently the integration directly into the Portal tab does not work.

CHAPTER

NINE

CONDITIONS REFERENCE

This chapter documents the *conditions* that the Guardian provides for configuring *capabilities* on *roles*. This is of interest for both *app developers* and *guardian admins*, that want to configure roles properly.

All conditions listed here are created in the `guardian` app's `builtin` *namespace*. Therefore the identifier of any condition is `guardian:builtin:condition_name`, where `condition_name` is the name of the specific condition.

Note: Requests to the *Authorization API* supply both an `old_target`, the state of the *target* before a change, and a `new_target`, the state of the target after the change.

In this document, conditions on the target apply only to the `old_target`.

`actor_does_not_have_role`

Parameter name	Value type
role	ROLE (string)

This condition applies if the *actor* does not have the *role* specified in the `role` parameter.

`no_targets`

This condition applies if the authorization request does not contain a specific *target*.

`only_if_param_result_true`

Parameter name	Value type
result	BOOLEAN

This condition is included for testing and debugging purposes only and should not be used.

`target_does_not_have_role`

Parameter name	Value type
role	ROLE (string)

This condition applies if the *target* does not have the *role* specified in the `role` parameter.

`target_does_not_have_role_in_same_context`

Parameter name	Value type
role	ROLE (string)

This condition applies if the *target* does not have the *role* specified in the `role` parameter with the same *context* as the *actor's* role currently being evaluated. For example, if the actor's role is `company:default:admin` in the

context `DEPARTMENT1` and the `role` parameter is `company:default:user`, this condition would apply as long as the target does not have the role `company:default:user` with the context `DEPARTMENT1`.

target_field_equals_actor_field

Parameter name	Value type
target_field	STRING
actor_field	STRING

This condition applies if the specified field of the *actor* and the specified field of the *target* have the same value.

target_field_equals_value

Parameter name	Value type
field	STRING
value	ANY

This condition applies if the specified `field` of the *target* has the same value as specified in the `value` parameter.

target_field_not_equals_value

Parameter name	Value type
field	STRING
value	ANY

This condition applies if the specified `field` of the *target* does not have the same value as specified in the `value` parameter.

target_has_role

Parameter name	Value type
role	ROLE (string)

This condition applies if the *target* has the *role* specified in the `role` parameter.

target_has_role_in_same_context

Parameter name	Value type
role	ROLE (string)

This condition applies if the *target* has the *role* specified in the `role` parameter with the same *context* as the *actor's* role currently being evaluated. If for example the actor's role is `company:default:admin` in the context `DEPARTMENT1` and the `role` parameter is `company:default:user`, this condition would apply as long as the target has the role `company:default:user` with the context `DEPARTMENT1`.

target_has_same_context

This condition applies if any of the *target's roles* have the same *context* as any of the *actor's roles*.

target_is_self

Parameter name	Value type
field	STRING

This condition applies if the *actor* and the *target* are the same. Per default this is decided by comparing their `id` attribute. If the `field` value is specified this field is used for identification instead.

CHAPTER

TEN

GLOSSARY

actor
: A user or machine account that wants to access a *target* in an *app* in some way. For example, a user actor may want to read the email of another target user.

app
: An application installed into a UCS system from the App Center, or a third-party service provider that integrates with the UCS system. Specifically, applications or service providers that integrate with the Guardian.

app developer
: A person, company, or organization that develops software that is used with a UCS system, that integrates with the Guardian. This includes UCS App Center applications, as well as third-party service providers using a service connector.

app infrastructure maintainer
: A person who installs and manages UCS systems.

authentication
: Confirmation of a user's identity. The Guardian does not handle authentication.

authorization
: Confirmation of the access that a user has. The Guardian's job is to handle authorization after a user is authenticated.

Authorization API
: A REST[21] interface that allows an *app* to authorize an *actor* to use features of the app.

capability
: One or more *permissions*, optionally combined with one or more *conditions* that are joined by either an "AND" or "OR" relationship.

condition
: A criterion under which a *permission* applies.

context
: An optional tag that modifies when a *role* applies.

guardian admin
: A user with the `guardian:builtin:super-admin` *role*, who can manage all aspects of the Guardian and any *app* using the Guardian, including *capabilities* for users and groups.

guardian app admin
: A user with a *role* ending in `app-admin`, who can manage most aspects of an *app*, including which *capabilities* a user has for that app.

Management API
: A REST[22] interface that allows an *app* or *guardian admin* to manage the Guardian.

Management UI
: A limited web interface that allows an *guardian admin* or *guardian app admin* to manage the Guardian.

[21] https://en.wikipedia.org/wiki/REST
[22] https://en.wikipedia.org/wiki/REST

namespace
A categorization of Guardian elements within an *app*. For example, an office suite might create an `email` namespace in which to store *roles* and *permissions* related to email.

permission
An action that an *actor* can take in a specific *app*.

role
A string assigned to a user group, or object in order to use a *capability*. In a UCS domain this is usually done in UDM and currently supported for user objects only.

target
A resource in an *app* that an *actor* wants to access. Used in determining which *permissions* an actor has.

CHAPTER

ELEVEN

CHANGELOGS

11.1 Authorization API

11.1.1 1.1.0 (2023-12-22)

- Remove obsolete App Center settings.
- Migrate docker image to UCS base image

11.1.2 1.0.0 (2023-12-11)

- Initial release.

11.2 Management API

11.2.1 1.1.0 (2023-12-22)

- Remove obsolete App Center settings.
- Rename App Center setting for Management API Keycloak client secret.
- Migrate docker image to UCS base image

11.2.2 1.0.0 (2023-12-11)

- Initial release.

11.3 Management UI

11.3.1 1.1.0 (2023-12-22)

- Remove obsolete App Center settings.
- Migrate docker image to UCS base image

11.3.2 1.0.0 (2023-12-11)

- Initial release.

11.4 Guardian Manual

11.4.1 1.1 (2023-12-22)

- Rename App Center setting for Management API Keycloak client secret.

11.4.2 1.0 (2023-12-22)

- Initial release.

Managing user permissions for a UCS system is difficult and time-consuming. Historically, it has required knowledge of access control lists (ACLs), and applications have usually hard-coded permissions to specific roles such as the Domain Admin.

The Guardian provides an alternative to this system, where *applications* can register user permissions, which UCS system administrators can then manage and organize in roles with an easy-to-use web interface. The *applications* in turn can then query the Guardian for authorization questions regarding specific *actors* and enforce app specific behavior in accordance with the administrators configuration.

For example, suppose that you run a business where you have a human resources department and an IT department. You want your human resources department to have different access to installed applications than your IT department. You may want to give permissions to the head of your IT department to manage email, while your vacation tracking application can only be managed by the head of HR.

The Guardian provides a convenient way to manage these permissions, for applications that support integration with the Guardian.

This manual explains how both UCS system administrators, as well as developers of applications for a UCS system, can use the Guardian to manage what users are allowed to do in applications.

CHAPTER

TWELVE

AUDIENCE FOR THIS MANUAL

There are three different audiences for the Guardian manual:

- *Guardian Administrators*
- *App Infrastructure Maintainers*
- *App Developers*

12.1 Guardian administrators

A guardian admin is a superuser who administers the Guardian once it has been installed, as well as managing *apps* that integrate with the Guardian. A *guardian app admin* is a subset of the guardian admin role, which has limited abilities to manage specific apps within the Guardian. Whenever this manual refers to an `admin`, this means either the superuser or a limited app admin.

Note: Not all applications installed through the Univention App Center support integration with the Guardian and can be managed through the Guardian. Please see the manual for your specific application to determine if it supports the Guardian.

This manual does not assume any specific technical knowledge for admins of the Guardian. When possible, all instructions use a web browser.

The chapter on the *Management UI* (page 21) is geared towards admins.

12.2 App infrastructure maintainers

An app infrastructure maintainer is someone who is responsible for installing and maintaining a UCS system and applications installed from the Univention App Center.

This manual assumes some technical knowledge for app infrastructure maintainers, such as the ability to use the command line and read log files.

The most relevant chapters for app infrastructure maintainers are:

- *Installation* (page 7)
- *Configuration* (page 11)
- *Troubleshooting* (page 17)

12.3 App developers

An app developer is a person, company, or organization who develops either applications that are installed through the Univention App Center, or a third-party external service provider that in some way connects to a UCS system to provide services to users within that system, for example, using the ID Connector[23].

An *app* is either an App Center application or a third-party external service provider, that integrates with the Guardian.

This manual presumes that app developers have high technical knowledge, including using a command line, writing code, and making calls to an API.

The most relevant chapters for app developers are:

- *Management API and Authorization API* (page 41)
- *Developer quick start* (page 45)
- *Conditions Reference* (page 61)

[23] https://docs.software-univention.de/ucsschool-id-connector/index.html

INDEX

R

T