# CatIS

## Installation and Configuration Guide

Version 24.1

# Table of Contents

## About this guide

This guide is meant for system administrators, who will be responsible for installation and technical configuration of the application and supporting software. This guide will not explain how to work with the application or how to define the configuration objects within the application, for that see the separate Administration Guide document.

The guide is last updated to CatIS version 24.1.

# Chapter 1: Installing CatIS

### *System requirements*

CatIS can be installed in a traditional way (components are installed and run in the operating system) or components can be installed into containers. It's also possible to mix the two methods, e.g. running MongoDB natively and running CatIS in a container or the opposite.

### *Native installation*

CatIS is based on MongoDB and Play Framework, if installing CatIS components natively, the operating system and hardware requirements depend on MongoDB and Play requirements.

**MongoDB (Community or Enterprise)**

CatIS supports MongoDB versions 4.0 up to 8.0

Check MongoDB supported operating systems and most recent installation instructions at: https://docs.mongodb.org/manual/installation/

Don't forget to configure MongoDB to auto-start on system startup.

**Java**

One of the Java LTS versions 11, 17, or 21. We recommend using at least Java 17.

To check that you have the corrent Java version, run: `java -version`

For default setup, install the database and application on the same server.

**Database installation**

Install MongoDB on the server with default options, please check up-to-date installation instructions for selected MongoDB version on supported platforms at https://docs.mongodb.org/manual/installation/

After installing and starting the MongoDB server, no additional setup tasks are required for the database.

**Application installation**

Unzip the distribution package on the target server and move/rename as you like

```
$ unzip catis-<version>.zip
```

In the following examples the <catis-dir> refers to the unzipped directory where CatIS files will be found.

The start script comes in two versions, a bash shell script, and a windows .bat script. For Unix users, zip files do not retain Unix file permissions, so when the file is expanded, the start script may be required to be set as an executable

```
$ chmod +x <catis-dir>/bin/catis
```

Then test-run the server with (`<secret>` is a random string of at least 32 characters):

```
$ <catis-dir>/bin/catis -Dplay.http.secret.key=<secret>
```

This will run the application on default port 9000 and context path 'catis' (to access it, the URL is

http://server:9000/catis). Verify that there are no error messages and the CatIS front page can be opened in the browser.

To exit the server, press Ctrl+C.

Catis comes with an eample systemd unit file `catis.service` that can be used for auto-starting CatIS on systemd-compatible servers, e.g. Ubuntu, CentOS, etc. Copy (or link) the file under `/etc/systemd/system/` and change the paths etc to match your environment. Then use `systemctl enable catis` and `systemctl start catis` commands to enable the service auto-start and start it.

If this is a small test configuration, you can use the built-in web server on the default port 9000 or configure it to another port by setting additional `http.port` parameter (on command line or in the catis.service file) like this

```
$ <catis-dir>/bin/catis -Dhttp.port=80 -
Dplay.http.secret.key=UsV23sY5qaqJcDC7ukkAv2ej8uUHhEQ2
```

Note that on Unix/Linux you probably need root privileges to bind to port 80.

On Windows, the start script has a .bat extension:

```
C:\catis\bin\catis.bat -Dhttp.port=80 -
Dplay.http.secret.key=UsV23sY5qaqJcDC7ukkAv2ej8uUHhEQ2
```

For a full description of usage invoke the start script with a `-h` option.

For production environments, it is recommended to use a separate front-end web server, e.g. Apache, Nginx, Caddy or similar.

### *Running in containers*

Install Docker Engine following the official documentation https://docs.docker.com/engine/install/

(Recommended) Create a directory for CatIS configration files, change working directory.

Create a new file with the name compose.yaml, example contents:

```
name: catis

services:
  catis:
    image: upmind/catis:latest
    restart: unless-stopped
    ports:
      - 9000:9000
    environment:
      # APP_CONTEXT: /catis
      MONGODB_URI: mongodb://mongo:27017/catis
      PLAY_APP_SECRET:
      AUTH_SIGNER_KEY:
      AUTH_CRYPTER_KEY:
    volumes:
      - ./application.conf:/app/conf/application.conf:ro
      - ./logs:/app/logs

  proxy:
```

```
    image: caddy:latest
    restart: unless-stopped
    ports:
      - 80:80
      - 443:443
    volumes:
      - ./Caddyfile:/etc/caddy/Caddyfile
      - caddy_data:/data
      - caddy_config:/config

  mongo:
    image: mongodb/mongodb-community-server:7.0.14-ubuntu2204
    restart: unless-stopped
    ports:
      - 127.0.0.1:27017:27017
    volumes:
      - mongo_data:/data/db

volumes:
  mongo_data:
  caddy_data:
  caddy_config:
```

To use Caddy's automatic TLS certificate management, make sure the external IP and hostname are registered properly in the DNS and ports 80 and 443 are accessible from the external Internet.

Create file named Caddyfile, example contents:

```
# The Caddyfile is an easy way to configure your Caddy web server.
#
# Unless the file starts with a global options block, the first
# uncommented line is always the address of your site.
#
# To use your own domain name (with automatic HTTPS), first make
# sure your domain's A/AAAA DNS records are properly pointed to
# this machine's public IP, then replace ":80" below with your
# domain name.

:80 {
    # when application context is set to "/"
    # reverse_proxy http://catis:9000

    # when application context is set to "/catis"
    reverse_proxy /catis* http://catis:9000
}

# Refer to the Caddy docs for more information:
# https://caddyserver.com/docs/caddyfile
```

Copy the applicaton.conf file into that directory or create a new file with that name, make necessary changes to the configuration. If upgrading from previous installation, the contents of this file should be what was before in prod.conf file (prod.conf is not used anymore).

Run the containers with `docker compose up`, check that there are no error messages and that

after some time, CatIS is accessible from the browser. If everything is ok, stop the containers with Ctrl-C and start them again in background with `docker compose up -d`

# Chapter 2: Configuration

### *Configuration files*

The default configuration is in the file `conf/reference.conf`. To change the defaults, the changes should be made in a separate configuration file `conf/application.conf` that then overrides the necessary parameters. This file with the specific changes should then be backed up for application upgrade in the future.

Then simply change the parameters to your environment, if necessay copy-paste the relevant parameter from the `reference.conf` first to the `application.conf` file.

If CatIS is run in a container, you can copy the `reference.conf` file from the container to inspect the default parameters.

### *Basic Configuration*

#### Application secret

Before you run the application in production mode, one thing you need to do is to generate an application secret. The application secret should be kept private, as anyone that can get access to the secret will be able to generate any session they please, effectively allowing them to log in to your system as any user.

The application secret is initally configured in `conf/reference.conf` with the property name `play.http.secret.key`, and defaults to `changeme`. As the default suggests, it must be changed for production (in the `appplication.conf` file). When started in production mode and the application finds that the secret is not set, or if it is set to `changeme`, or is too short, it will throw an error and the server will not start.

It is recommended that you do not put the production application secret directly in configuration files that are stored in backup sets, rather it should be configured only on your production server.

The recommended method is to place the application secret in an environment variable. In this case, place the following configuration in your `application.conf` file instead of the value itself:
`play.http.secret.key=${?PLAY_APP_SECRET}`

This configuration sets the secret to come from an environment variable called `PLAY_APP_SECRET` if such an environment variable is set.

#### Application Context Path

Application context path is the part of the application URL after the hostname (and port). Default installation comes with context path configured as '/catis', but you can change it to suit your environment, for example if the application should be at the root of the server, change this to "/". The default setting is in the `reference.conf`

```
play.http.context = "/catis"
```

When changing the application context, you should also adjust the reverse proxy configuration to match.

### URL prefix in links

In order to generate proper working links to the application, e.g. in a workflow notification email, change the configuration properties for URL prefix according to your environment (more about the certprefix parameter in section "Certificate Authentication"):

```
play.http.urlprefix = "https://catis.example.com"
play.http.certprefix = "https://catis.example.com"
```

### Authentication domain

For the authentication to work properly, especially over https connections, the authentication cookie domain must match the (external) host name or domain of the server:

```
silhouette.authenticator.cookieDomain="example.com"
```

NB! If you find that the authentication does not work when the cookie domain is set to a domain name (or works in some browsers but not in others), you must use a subdomain or host name instead, otherwise the authentication cookie cannot be set. For an explanation, see: https://publicsuffix.org/learn/

### Authentication keys

You should also change the authentication crypter and signer keys that are used by the authentication module, like you did for the application key, e.g.

```
silhouette.authenticator.signer.key =
Eme2B6hpGKgpbTph3FW4KMKcukahfL4K
silhouette.authenticator.crypter.key =
pZhZZc8na8tSGFJxVcXuw5zjKFkcrwKm
```

### Authentication methods

CatIS supports several authentication mechanisms that can be independently enabled or disabled via a configuration parameter:

```
authentication.methods = ["password", "certificate"]
```

To disable for example password authentication, remove the word "password" from the list, or simply rename it to something else, e.g.

```
authentication.methods = ["#password", "certificate"]
```

will disable password authentication (and signup by email and password).

Similarily, to disable certificate authentication, remove the word "certificate" from the list.

### User self-registration

To enable or disable user self-registration (after the initial admin user has registered), change the configuration parameter to true (self-registration is enabled) or false (new user accounts need to be registered by the admin or organisation user):

```
selfregistration.enabled = false
```

### New institution registration requests

To enable or disable anonymous users to submit requests to register a new institution in CatIS, change the configuration parameter to true or false (defaults to true):

```
anonymous.registration.enabled = true
```

To enable or disable authenticated users (but who are not admins) to submit requests to register a new institution in CatIS, change the following configuration parameter to true or false (defaults to true):

```
authenticated.registration.enabled = true
```

### Allowed hosts

CatIS (or more precisely, the underlying Play framework) provides a filter that lets you configure which hostnames can be used to access the application. This can help protect from certain types of attacks. The filter is imply a whitelist of allowed hosts, that should match your server name. The following example allows requests to example.com, its subdomains, and localhost:9000.

```
play.filters.hosts {
  allowed = [".example.com", "localhost:9000"]
}
```

### Database connection

Default installation assumes that the MongoDB server is installed on the same host and with default settings. If you have installed the MongoDB server on another host or listening on a different port, you need to chage the database configuration parameters in the `application.conf` file:

```
mongodb.uri = "mongodb://localhost:27017/catis"
```

NB! When running CatIS in a container, use the MongoDB container name as hostname instead of localhost, e.g.

```
mongodb.uri = "mongodb://mongo:27017/catis"
```

### Mail server settings

Some actions in CatIS cause notifications to be sent out via e-mail. For this to work, you need to define the proper SMTP settings in the `application.conf` section `play.mailer`, at least `host` and `from` and `registration.sendto` (the address used for notifications about new registration requests)

### Format of generated identifiers

When creating new objects (institutions, systems, etc) the application assigns each a unique identifier based on the configurable format. If the defaults need to be changed, you can change them in the `application.conf`. The interpretation of the formatting patterns is described in

java.util.Formatter javadocs, a sequential number is passed in as int parameter. The defaults are:

```
institution.idformat = "IN%05d"
system.idformat = "IS%05d"
service.idformat = "SE%05d"
publicservice.idformat = "PS%05d"
asset.idformat = "IA%05d"
registration.idformat = "RE%05d"
```

### Workflow reminders

The application sends out reminder emails for overdue workflow actions. By default, the reminders are sent at 6:00 every morning, for 3 consecutive days. To change the time of reminders, change the configuration parameter

```
akka.quartz.schedules.WorkflowReminder.expression = "0 0 6 * * ?"
```

See the following link for the syntax: https://github.com/enragedginger/akka-quartz-scheduler#schedule-configuration

To change the number of times the reminders are sent out, change the parameter:

```
workflow.remind.days = 3
```

### Access log settings

CatIS logs user access to the application, i.e. reads, updates, creation or delettion of the objects in the application. After a certain time period, the old logs are automatically deleted, you can configure the needed retention period in prod.conf

```
accesslog.ttl = 7.days
```

Creation, modification and deletion of the objects is always logged. If you are not interested in who reads what object and don't want that information to appear in the logs, you can disable read logs

```
accesslog.logReads = true
```

or you can disable log reads for just anonymous users, but keep logging authenticated users

```
accesslog.logAnonymousReads = false
```

#### *Additional configuration*

CatIS is based on Play Framework and there are a number of different configuration options that you can set in addition to the ones provided above. For detailed documentation about all the possible configuration values please see:
https://www.playframework.com/documentation/2.9.x/ProductionConfiguration

# Chapter 3: Languages and translations

### *Specifying supported languages*

CatIS supports several languages out of the box. The supported languages are specified in the `conf/application.conf` file and can be overriden in `prod.conf`, if you want to remove languages that are not relevant to your users or add new languages that are not included in the default package.

A valid language code is specified by a valid ISO 639-2 language code, optionally followed by a valid ISO 3166-1 alpha-2 country code, such as `fr` or `en-US`:

```
play.i18n.langs = [ "en", "en-US", "fr" ]
```

### *Disabling multi-language support*

The available languages appear in the language drop-down lists for users and also on the forms/views/etc configuration screens. If there is no need have the application in multiple languages, it is possible to remove these drop-downs and use the application in single language only. To disable multi-language support altogether, simply leave the language list empty:

```
play.i18n.langs = []
```

### *Language files*

Externalized strings (that are not modifiable via browser interface), are in the `conf/messages.xxx` files.

The default `conf/messages` file matches all languages. If you add a new language, you should also provide a language-specific message files such as `conf/messages.fr` or `conf/messages.en-US`

The structure of the language files is simple name-value pairs, for example:

```
first.name = First name
last.name = Last name
full.name = Full name

...
```

# Chapter 4. Setting up client certificate authentication

If client certificate authentication is desired, some special configuration is needed. First, a front-end web server is necessary that does the actual certificate requesting and validation. The web server needs to have a TLS certificate and any CA certificates configured. 2 configurations are tested, but the same principles could be applied to other front-end web servers.

### *Apache httpd*

An example configuration directives (based on the default httpd.conf) to set up Apache as a reverse proxy server and require client certificate authentication for certain paths.

The location of the configuration files can vary based on the distribution, change accordingly. The following is an example of the required directives.

In the conf/httpd.conf file:

```
# For setting request headers
LoadModule headers_module modules/mod_headers.so
# For reverse proxy support
LoadModule proxy_module modules/mod_proxy.so
LoadModule proxy_http_module modules/mod_proxy_http.so
# SSL support
LoadModule ssl_module modules/mod_ssl.so
<IfModule ssl_module>
Include conf/extra/httpd-ssl.conf
SSLRandomSeed startup builtin
SSLRandomSeed connect builtin
</IfModule>
```

In the conf/extra/httpd-ssl.conf file:

```
<VirtualHost *:443>
  SSLEngine on
  ServerName catis.example.com:443
  SSLVerifyClient none

  SSLCertificateFile "${SRVROOT}/conf/ssl/server.crt"
  SSLCertificateKeyFile "${SRVROOT}/conf/ssl/server.key"
  SSLCACertificateFile "${SRVROOT}/conf/ssl/ca.crt"
  SSLVerifyDepth 5
  SSLProtocol -all +TLSv1.2
  <Location "/catis/certificate">
     SSLVerifyClient require
     SSLVerifyDepth 5
     SSLOptions +StdEnvVars
     RequestHeader set SSL_CLIENT_VERIFY %{SSL_CLIENT_VERIFY}s
     RequestHeader set SSL_CLIENT_S_DN %{SSL_CLIENT_S_DN}s
     RequestHeader set SSL_CLIENT_CERT %{SSL_CLIENT_CERT}s
  </Location>
  ProxyPreserveHost On
  ProxyPass / http://127.0.0.1:9000/
  ProxyPassReverse / http://127.0.0.1:9000/
</VirtualHost>
```

The conf/ssl/server.crt and conf/ssl/server.key should contain the server's TLS certificate and key, the conf/ssl/ca.crt must contain the CA certificates chain, that issued the client certificates.

**NB!** TLS1.3 dropped support for dynamic TLS renegotiation. To use this configuration Apache must be set to TLS1.2 (the SSLProtocol directive above). If TLS1.3 is desired, configuration with separate hostnames (similar to Nginx below) must be used. It is also possible, that future versions of Apache and/or web browsers drop support for renegotiation altogether.

### *Nginx*

As Nginx server does not support dynamic TLS renegotiation, additional steps are needed.

To support path-based client certificates, another hostname must be registered in the DNS for the site that requires client certificate. Also, the server's TLS certificate should have both hostnames in the subjectAltName attibute.

Example configuration directives for Nginx:

```
http {
  server {
    listen       443 ssl;
    server_name  catis.example.com;
    ssl_certificate      server.crt;
    ssl_certificate_key  server.key;
    ssl_client_certificate   ca.crt;
    ssl_verify_client off;
    ssl_session_cache    shared:SSL:1m;
    ssl_session_timeout  5m;
    ssl_ciphers  HIGH:!aNULL:!MD5;
    ssl_prefer_server_ciphers  on;
    location /catis {
      proxy_pass http://127.0.0.1:9000/catis;
      proxy_set_header Host $host;
      proxy_set_header X-Forwarded-For $remote_addr;
      client_max_body_size 100M;
    }
  }

  server {
    listen       443 ssl;
    server_name  id.example.com;
    ssl_certificate      server.crt;
    ssl_certificate_key  server.key;
    ssl_client_certificate   ca.crt;
    ssl_verify_client on;
    ssl_session_cache    shared:SSL:1m;
    ssl_session_timeout  5m;
    ssl_ciphers  HIGH:!aNULL:!MD5;
    ssl_prefer_server_ciphers  on;
    location /catis/certificate {
      proxy_pass http://127.0.0.1:9000/catis/certificate;
      proxy_set_header Host $host;
      proxy_set_header X-Forwarded-For $remote_addr;
      proxy_set_header SSL_CLIENT_VERIFY $ssl_client_verify;
```

```
      proxy_set_header SSL_CLIENT_S_DN $ssl_client_s_dn;
      proxy_set_header SSL_CLIENT_CERT $ssl_client_cert;
      client_max_body_size 10M;
    }
  }
}
```

## Chapter 5. Connecting to XRoad infrastructure

If you have access to XRoad, you can configure CatIS to connect to the XRoad security server and automatically retrieve service WSDL-s for services, that are registered in CatIS as XRoad services. The following configuration settings specify the connection parameters to the XRoad security server and registered organisation data (the organisation who manages CatIS). Consult with your XRoad administrator for the proper values

Address or hostname of the security server
`xroad.server.address = "http://xroad.example.com"`

Endpoint URL of the security server
`xroad.server.endpointUrl = "http://xroad.example.com/cgi-bin/consumer-proxy"`

`xroad.instance = ee-test`
XRoad instance code

`xroad.client.memberClass = COM`
XRoad member class of the organisation

`xroad.client.memberCode = 12345678`
XRoad member code (registration code) of the organisation

`xroad.client.subsystemCode = catis`
Registered subsystem code of CatIS

`xroad.client.userId = 123456789`
Identification code of the user, who is authorising the xroad requests

CatIS will display a link to the service WSDL for the service objects, that are marked as XRoad services. As this information is held in a field, that can be dynamically configured on the service definition form, to be able to recognise XRoad services, you need to tell CatIS what value in what field indicates an XRoad service
`xroad.service.fieldname = "service.general.classification"`
`xroad.service.fieldvalue = "X-Road"`

For retrieving service WSDL-s, CatIS needs to know the member class of the owning intitution. As this information is kept in a field that can be dynamically configured, you need to tell here, what field is holding that value in the institution form.
`xroad.memberclass.fieldname = "class"`

# Chapter 6. Customizing the user interface

### *Setting enabled modules*

Choosing what modules are visible in the user interface can be done in two different ways:
1) explicitly naming all enabled modules by adding a configuration entry with the key "catis.modules.enabled" and a string list of modules to be enabled, e.g.

```
catis.modules.enabled = ["publicservices", "institutions",
"systems", ...]
```

Default setting with all modules enabled is in the reference.conf.

2) naming disabled modules – this is a safer way if the goal is to remove some modules that are not in use (yet), for example workflows. This can be done by adding an entry to the configuration file with the key "catis.modules.disabled" and a string list of modules to be disabled, e.g.

```
catis.modules.disabled = ["assets", "processes"]
```

will remove "Assets" from the top navigation and additionally will remove Workflows from all the pages.
Before adjusting the modules, it is strongly recommended to have a good understanding of what the modules do and how they are related to each other. For example, disabling "users" module will remove the possibility to assign user permissions to the various objects and "institutions" module is a prerequisite for all the other modules, etc.

### *Using custom CSS design*

If a file named `theme.css` exists a in `public/stylesheets/` directory under the CatIS installation directory, this file will be automatically included as a CSS stylesheet in all application pages.
To customize the look of the application, create a file `public/stylesheets/theme.css` and place own CSS style overrides in this file. Custom images should go to public/images/ and can be referenced in custom CSS.
There is also a file named `default-theme.css` in the same directory, that includes the default design. This file can be safely removed or taken as a starting point for own custom theme.css desing.