Version
0.2

Status
Approved

Approver

Author
Daniel Joachim Larsen

KOMBIT

AULA – WIDGET LOCAL DEVELOPMENT

T0150 – Widget Local Development

Document history

Version	Date	Author	Status	Comments
0.1	22-06-2023	Daniel Joachim Larsen	Draft	
0.2	23-06-2023	Andreas Laursen	Approved	

References

Reference	Title	Author	Version
[WidgetGuide]	T0150 – Widget guide	Lasse Poulsen	0.1

Table of contents

1 Background

To make Aula successful as the communication platform for all parties involved in schools and daycare institutions, the system will be able to both

- Display user interfaces from other systems alongside the relevant communication tools within Aula
- Perform single-sign-on on web links into other systems, enabling automatic login in with the end-users UNI-Login Id

Such user interfaces and single-sign-on links from other systems are referred to as Widgets, and this guide describes how a Widget Supplier can develop & test Widgets to be used in Aula.

Examples of possible Aula Widgets:

- Widgets from the individual Municipalities' Learning Platforms (e.g. Homework or links to digital learning tools)
- Widget from absence registration solution to register student absence.
- Single-sign-on links that can be placed as a toolbox on relevant Aula Dashboards

1.1 Purpose of this document

Testing that a widget works with Aula is, of course, necessary ahead of deployment. Classically, the way to test a widget was to use the Widget previewer as described in [WidgetGuide]. With the addition of the Local Development Environment there is now an alternative way of testing widgets.

This document serves the purpose of providing instructions to Widget suppliers on setting up, testing, and deploying their widgets using the local development environment. This document is specifically designed for widget developers responsible for creating and updating widgets for Aula within the supplier's organization, the document aims to guide them through the necessary steps of developing widgets for Aula locally.

1.1.1 Relation to Widget previewer

Given the enhancement of the local widget development environment, which now supports multiple widget instances and the inclusion of test users, it is advised that widget suppliers utilize this environment for their development needs.

1.2 Target group

This guide is primarily intended for developers responsible for developing Widgets for Aula. Readers are assumed to have knowledge of web development, in particular HTML, JavaScript, CSS and REST Web API's.

In addition, the guide describes the process of setting up a local development environment for developing Aula Widgets.

2 Setup

This section describes the technical details for setting up local development of Widgets for Aula. If you are not yet an Authorized Widget Supplier, refer to [WidgetGuide] section 3.1 to become a Widget Supplier. Note that to gain access to the resources below it is necessary to subsequently create a case on this in the toolkit

2.1 Installation

To ensure that Widget Suppliers can run the development environment locally. The Widget Supplier must download and install Node JS and Docker on their machines.

Node.js is utilized for hosting the frontend of the local Widget development environment.

Docker is used to host the backend API, which generates the Aula token and lists all the test users.

Installing Node JS:

1. Go to the official Node JS website (https://nodejs.org/) and download the recommended version for your operating system.
2. Follow the installation instructions for your operating system.
3. Verify that Node JS is installed by opening a terminal or command prompt and typing "node -v". If Node JS is installed properly, it should display the version number.

Installing Docker:

1. Go to the official Docker website (https://www.docker.com/) and download the appropriate version for your operating system.
2. Follow the installation instructions for your operating system.
3. After the installation is complete, verify that Docker is installed by opening a terminal or command prompt and typing "docker --version". If Docker is installed properly, it should display the version number.

To get the development environment up and running. Refer to the instructions described in the README.md file.

2.2 Git

This section describes the technical details of cloning and maintaining the local widget development repository.

2.2.1 Clone the project

Go to the Azure DevOps and clone the WidgetDevelopmentEnvironment repository.

2.2.2 Remote origins

To ensure continuous updates to the cloned project, assign a new remote origin name to the aula Azure DevOps repository.

Example

```
git remote rename origin aula
```

The widget supplier should add their own remote origin to push their changes towards.

```
git remote add origin {url}
```

Set the upstream to the new origin

```
git branch --set-upstream-to origin/master
```

2.2.3 Updates

All Widget Suppliers will receive an Email when the WidgetDevelopmentEnvironment repository is updated.

Typically, an update will occur when a new release update of Aula is up.

Regular updates are always cosmetic, in case there are more substantial changes, an email notice will be sent out to all widget suppliers regarding those changes.

After the Widget Supplier have received the Email notification, the widget supplier can update their copy of the repository with the following command
Example:

```
git pull aula master # pulls changes from the remote origin aula
```

3 Widget development

This Section describes the technical details of developing a widget for Aula.

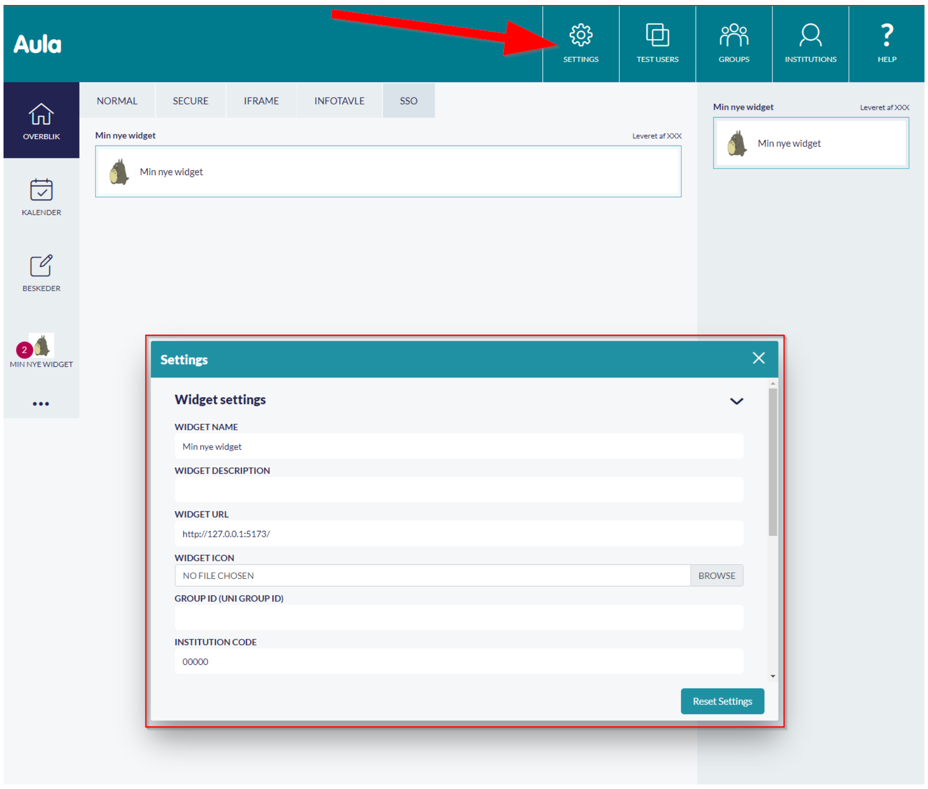
3.1 Getting Started

Widget suppliers can use their preferred IDE to develop the widget source code. All code examples will be illustrated in the Jetbrains IDE.

To understand the technologies used in Aula, refer to [WidgetGuide] section 4 Technical guide.

3.2 SSO Widgets

To begin configuring SSO widgets the widget supplier can start the development environment and configure the SSO widget in the settings menu by setting the name, URL and icon, the SSO widget will be ready and configured.



Settings

Widget settings

WIDGET NAME
Min nye widget

WIDGET DESCRIPTION

WIDGET URL
http://127.0.0.1:5173/

WIDGET ICON
NO FILE CHOSEN BROWSE

GROUP ID (UNI GROUP ID)

INSTITUTION CODE
00000

Reset Settings

3.3 Iframe Widgets

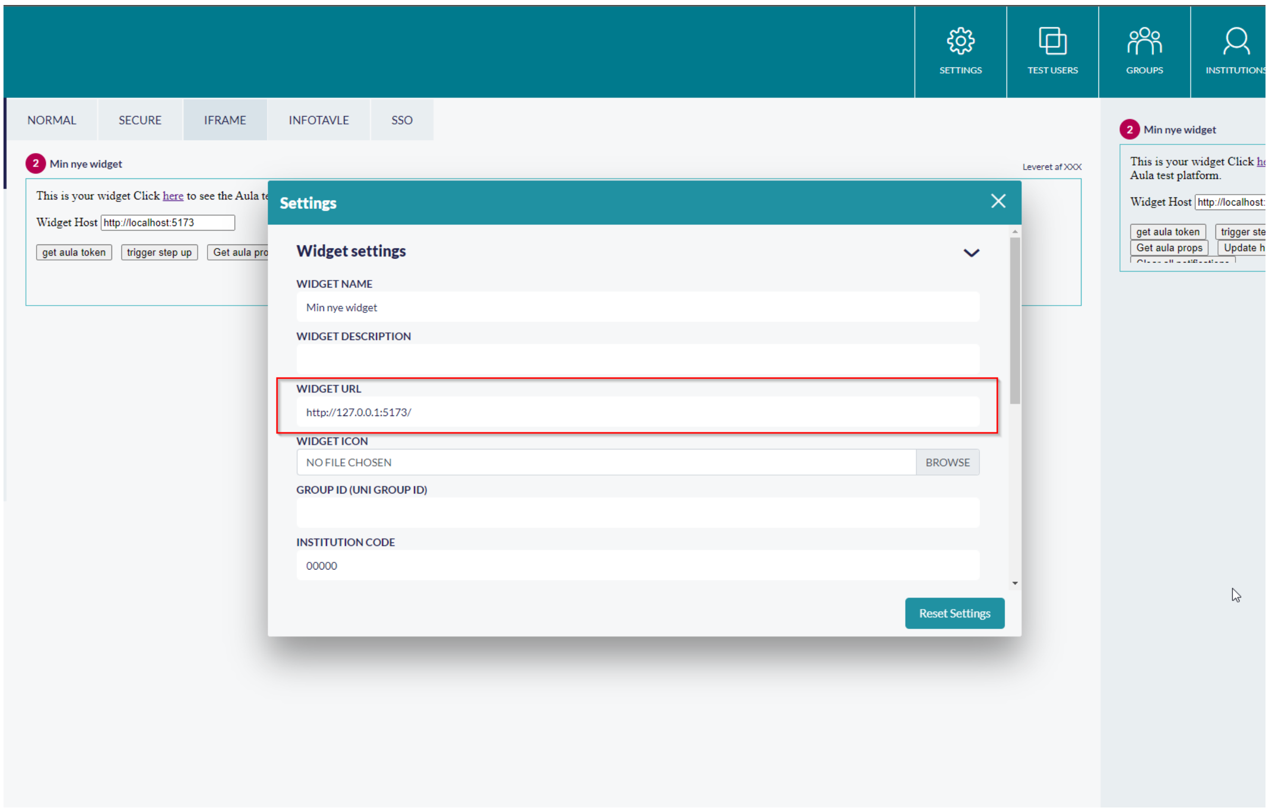
When developing iframe widgets. It is recommended that the Widget supplier create a separate project in their preferred language or Javascript framework that the Widget development environment can communicate with.

There is a sample setup in the Widget development environment at `src/IframeWidget` Aula recommends that this folder is treated as an example for how to setup a Iframe widget, however widget suppliers can also develop in the IframeWidget folder as well.

An Iframe widget will be treated the same as a secure widget. Refer to [WidgetGuide] section 4.2 for information about available parameters for iframe widgets.

In the settings menu, the widget supplier can point towards the widget's hosted URL by setting the Widget URL.

Example



Settings

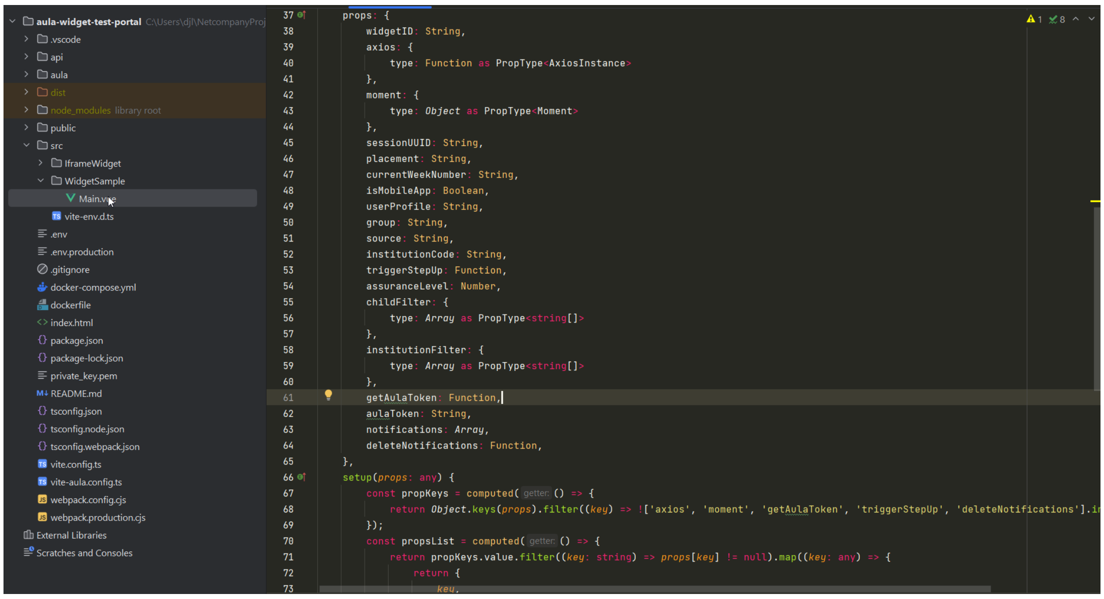
Widget settings

WIDGET NAME	Min nye widget
WIDGET DESCRIPTION	
WIDGET URL	http://127.0.0.1:5173/
WIDGET ICON	NO FILE CHOSEN — BROWSE
GROUP ID (UNI GROUP ID)	
INSTITUTION CODE	00000

Reset Settings

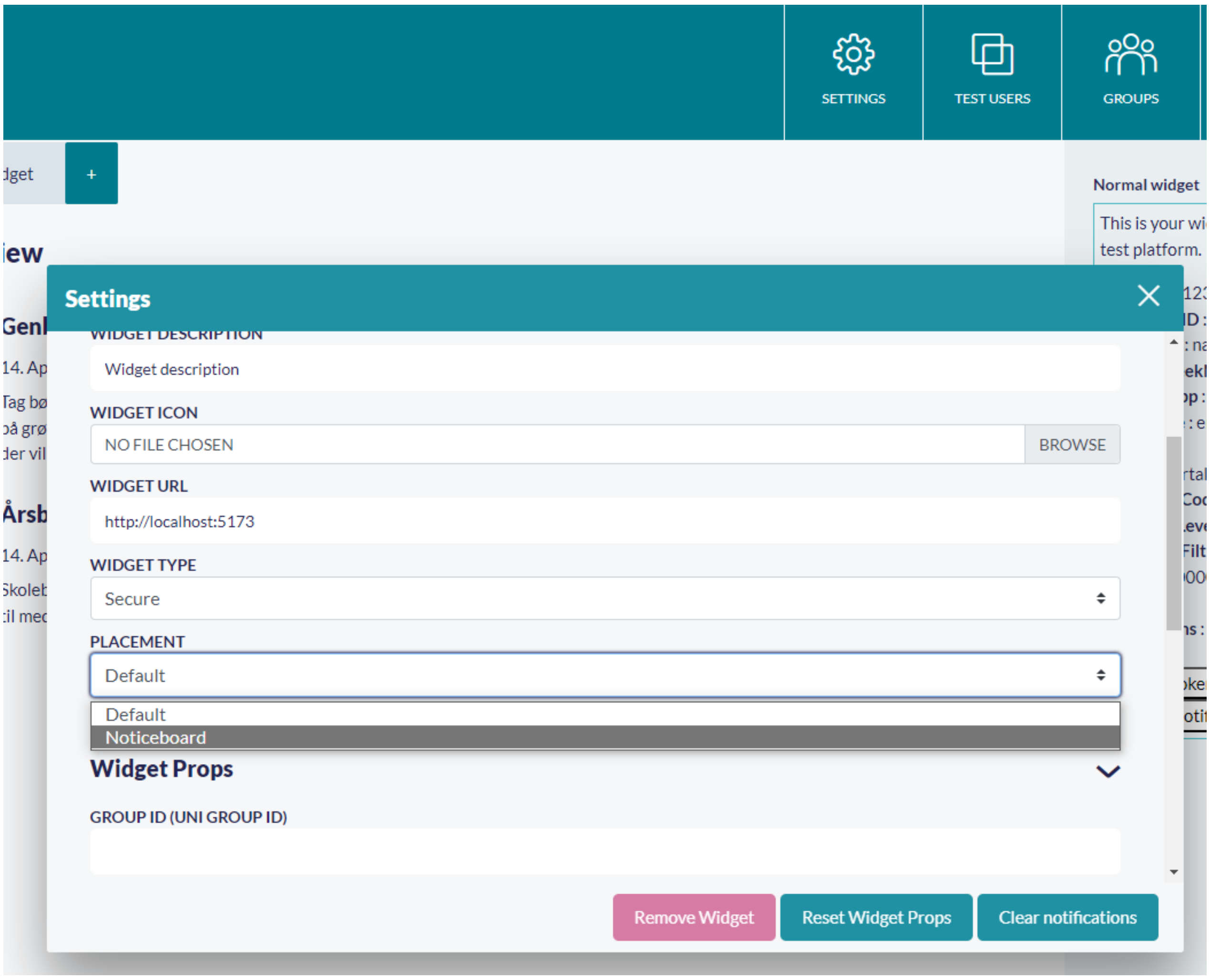
3.4 Other Widgets

There is an example of how-to setup a widget for Widget vendors in `src/WidgetSample`. This folder is only meant to serve as a template example, any custom widgets should be created in a separate directory fx. `src/MyCustomWidget/Main.vue`.



```
    props: {
        widgetID: String,
        axios: {
            type: Function as PropType<AxiosInstance>
        },
        moment: {
            type: Object as PropType<Moment>
        },
        sessionUUID: String,
        placement: String,
        currentWeekNumber: String,
        isMobileApp: Boolean,
        userProfile: String,
        group: String,
        source: String,
        institutionCode: String,
        triggerStepUp: Function,
        assuranceLevel: Number,
        childFilter: {
            type: Array as PropType<string[]>
        },
        institutionFilter: {
            type: Array as PropType<string[]>
        },
        getAulaToken: Function,
        aulaToken: String,
        notifications: Array,
        deleteNotifications: Function,
    },
    setup(props: any) {
        const propKeys = computed( () => {
            return Object.keys(props).filter((key) => !['axios', 'moment', 'getAulaToken', 'triggerStepUp', 'deleteNotifications'].i
        });
        const propsList = computed( () => {
            return propKeys.value.filter((key: string) => props[key] != null).map((key: any) => {
                return {
                    key,
```

Customizing the widget type and placement can be done through settings.



WIDGET DESCRIPTION
Widget description

WIDGET ICON
NO FILE CHOSEN BROWSE

WIDGET URL
http://localhost:5173

WIDGET TYPE
Secure

PLACEMENT
Default
Default
Noticeboard

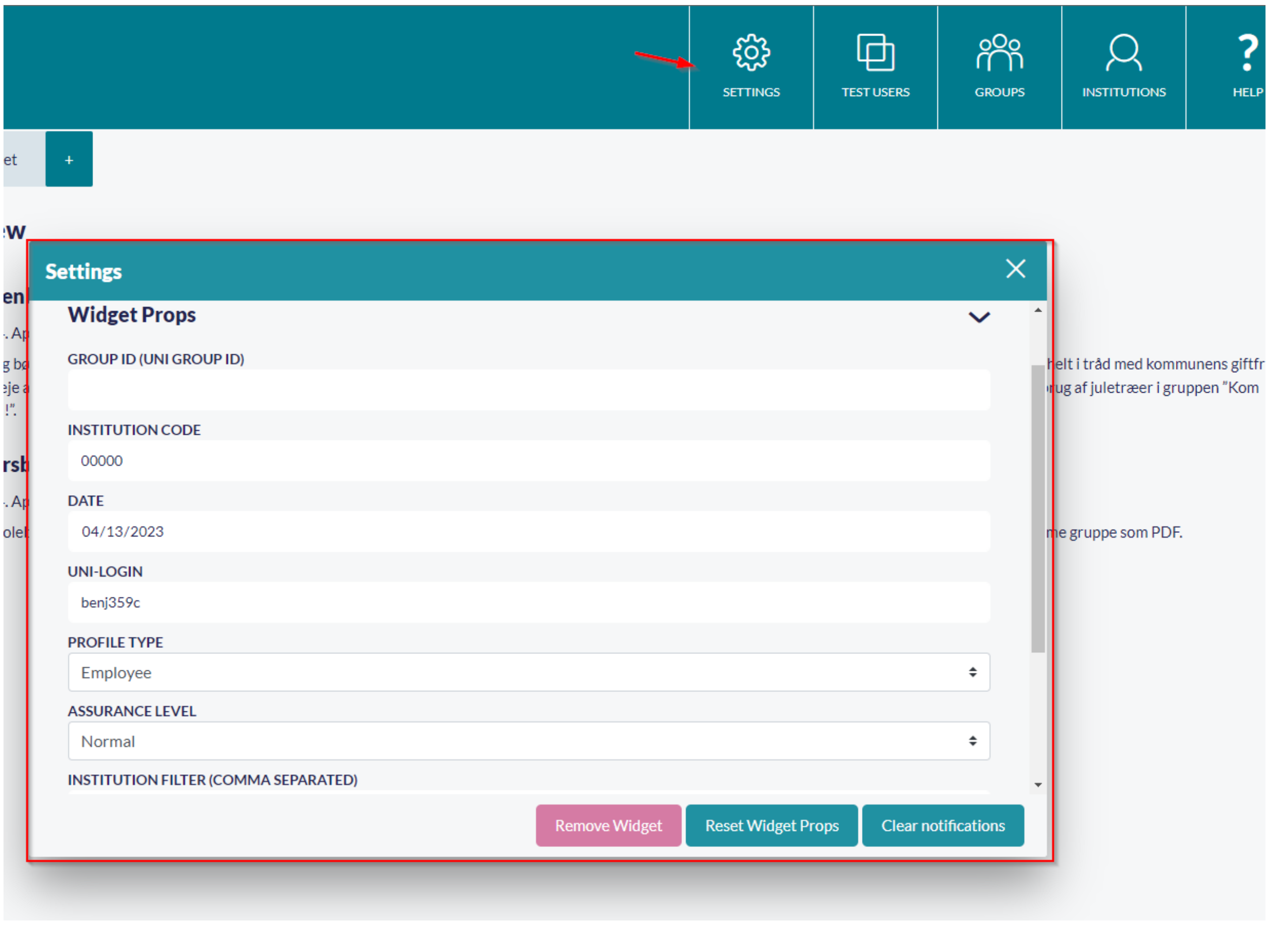
Widget Props

GROUP ID (UNI GROUP ID)

3.5 Widget props

Refer to [WidgetGuide] section 4.2 for information about available widget props for any widget types.

A widget supplier can manipulate props sent to a widget through the settings button in the top menu.



Settings

Widget Props

GROUP ID (UNI GROUP ID)

INSTITUTION CODE
00000

DATE
04/13/2023

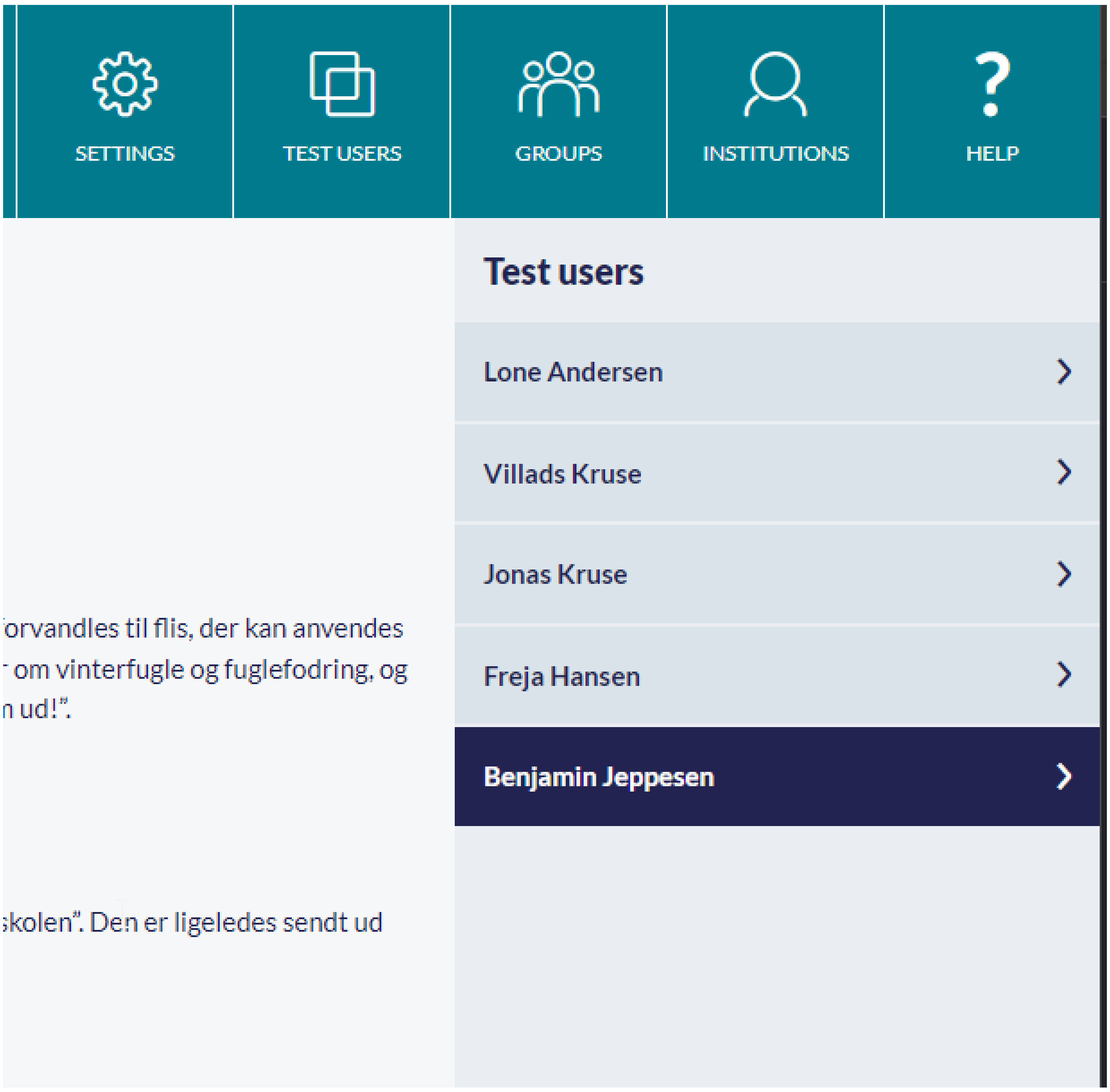
UNI-LOGIN
benj359c

PROFILE TYPE
Employee

ASSURANCE LEVEL
Normal

INSTITUTION FILTER (COMMA SEPARATED)

Alternatively, widget props can be setup for specific purposes by choosing a test user in the top menu.



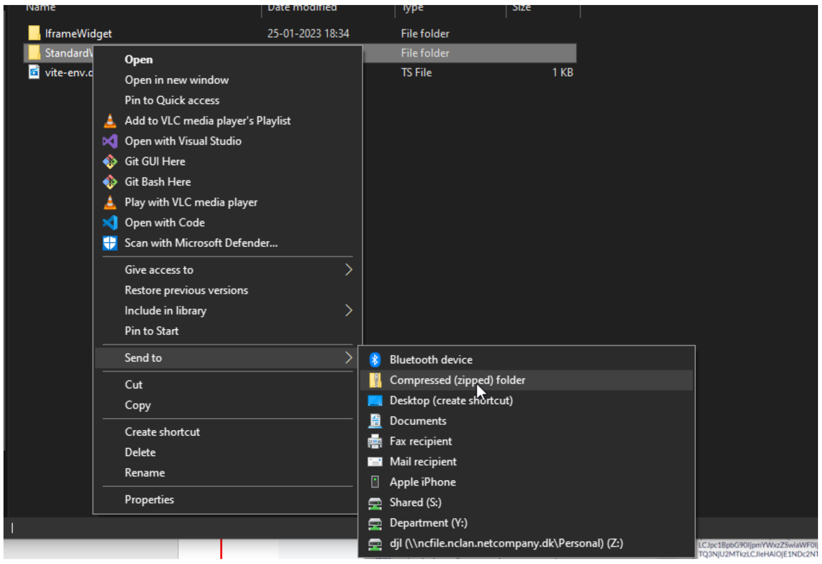
NOTE: To use test users, the docker container in the widget development environment must be running.

4 Toolkit

Toolkit works as the central unit for anything widget related. This page will contain a document library with guides and documentation, a Cases page to create applications for new/updating/deleting widget, incidents and other service requests and the Widgets source code.

4.1 Compress Widget

When a Widget suppliers widget is ready, compress the folder to a zip file. In Windows this can be done by right clicking the folder and compress to zip file.



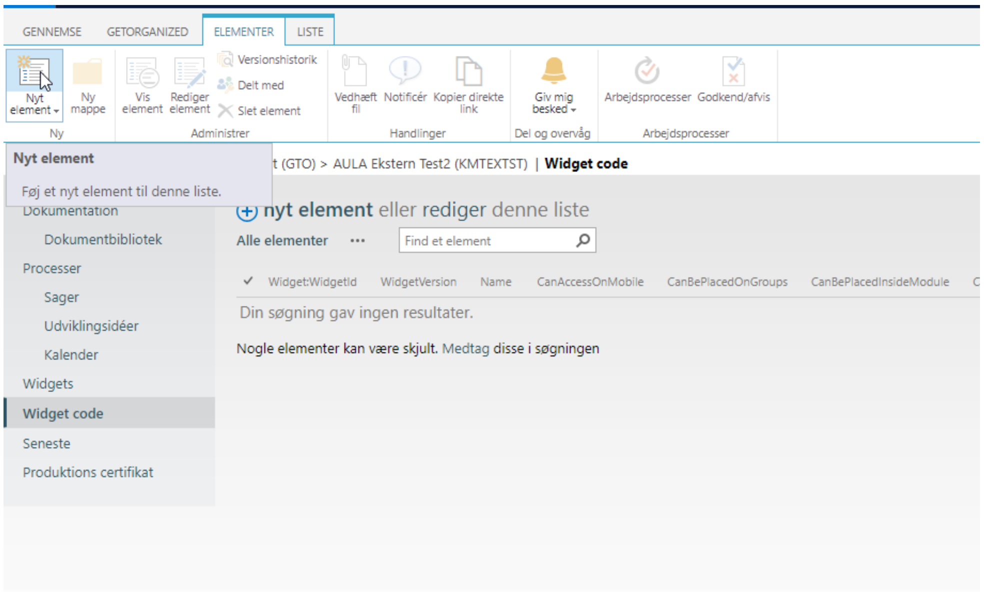
4.2 Applying for new or change widget

Refer to [WidgetGuide] section 4.3 for general information on how to add new widgets in Toolkit.

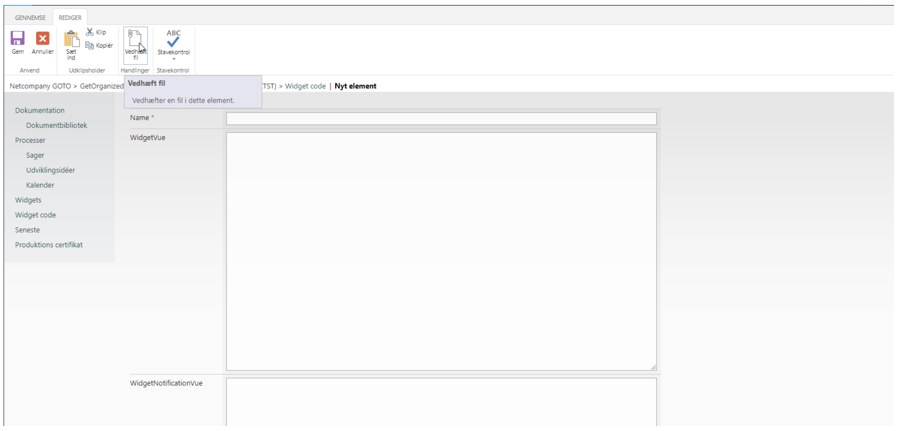
4.3 Review Widget Code

Widget Code can be uploaded in the Widget Code menu.

At the top of this menu page, the upload can be performed by clicking a new element (Nyt element)



Attach the Widget zip file.



5 Limitations

You should read all limitations in the [WidgetGuide], however, the listed limitations here are specifically relevant for when using the local development environment.

5.1 No use of v-html

The v-html directive in Vue.js allows you to render dynamic HTML content on your web page by binding a string of HTML to an element's innerHTML. While v-html can be a useful feature, it can also introduce a security risk if used improperly.

v-html can allow an attacker to inject malicious code into your web page, leading to a potential XSS attack. If the data that you're binding to v-html is not properly sanitized or validated, an attacker can inject script tags or other HTML elements that execute malicious code in the context of your page.

If you must render content Aula has made a custom component that sanitizes your input.

Example

```
<widget-html :html="strHtml"></widget-html>
```

****NOTE**: Links and inline javascript will be stripped from the content.

5.2 Usage of the aula folder

The Aula folder is for Aula developers only any changes in this folder will result in future merge conflicts when updating your cloned repo.

Aula recommends that you do not alter or make use of anything in the aula folder.

5.3 Imports

For non- iframe widgets, imports are limited to the following dependencies:

- element-ui
- bootstrap-vue
- lodash
- momentJS

- vue