netcompany

Version
1.10

Status
Approved

Approver
Andreas Laursen

Author
Lasse Poulsen

KOMBIT

AULA

T0150 – Widget guide

Document revisions

Version	Date	Author	Status	Remarks
0.1	05-12-2017	Lasse Poulsen	Draft	Initial version, based on T0150 – Widget guide – Stage 1
0.2	07-12-2017	Lasse Poulsen	Ready for review	Revised document structure and updated content with design from Stage 2.1.
0.3	13-12-2017	Lasse Poulsen	Ready for review	Review comments from ERHA addressed.
0.4	14-12-2017	Lasse Poulsen	Ready for review	Minor clarifications.
0.5	14-03-2018	Klaus Ulrik Bjerg	Ready for review	Further clarifications: getAulaToken, REST helper api, code examples. Cross-references to Design Guide.
1.0	10-04-2018	Benjamin Blankholm	Approved	Version changed to 1.0
1.1	23-11-2018	Morten Korsholm Terndrup	Ready for review	Added security limitations.
1.2	06-12-2018	Kasper Langgaard	Ready for review	Added toolkit guide 4.3 and widget variables overview 4.3.2
1.3	16-01-2019	Dong Duong Nguyen	Ready for review	Added contact info in “Become Widget Supplier” 3.1 and added screenshots for filters in Input parameters 4.2
1.3	19-07-2019	Jesper Lund Andersen	Ready for review	Answered last review comments.
1.3	18-09-2019	Jesper Lund Andersen	Ready for review	Revised sections 2.3.1.3 and 4.5 about widget notifications.
1.4	14-10-2019	Morten Bernhard Riis	Ready for review	SSO links will be send by GET on apps
1.5	04-12-2019	Morten Bernhard Riis	Ready for review	SSO links added parameters
1.6	14-04-2020	Malte Darling Dueholm	Approved	Updated KOMBIT E-mails on behalf of Erling Hansen
1.7	14-09-2020	Casper Slynge	Ready for review	Added notice board institution code and Element UI library

1.8	15-10-2020	Caspar Oreskov	Ready for review	Remove link to Vue documentation where example using “template” is suggested.
1.9	16-03-2021	Casper Skjærris	Ready for review	Update documentation of provided properties and SSO shortcuts to reflect OTP changes.
1.10	11-11-2021	Caspar Oreskov	Ready for review	Add status “Solved” and “Awaiting customer” in section 4.3.1 Removed status “Duplicated” in section 4.3.1
1.11	21-01-2022	Caspar Oreskov	Ready for review	Add Noticeboard widget type.
1.12	02-03-2022	Leslie Mielby	Ready for review	Add description of triggerStepUp props function in section 4.2.
1.13	20-02-2023	Daniel Joachim Larsen	Ready for review	Removed description of widget notifications not being supported in Aula. Added notifications and deleteNotifications rows in input parameters Updated section 4.5 notifications and deleteNotifications
1.14	29-03-2023	Andi Rosengreen Kjærsig Aaes	Ready for review	Updated general description of Widget notifications, and added new screenshots, Section 2.3.1.3 Added description of Technical guide for Widget notifications when sending notications to the user, Section 4.5.1
1.15	20-04-2023	Daniel Joachim Larsen	Ready for review	Added iframe widget description to section 2.3.1.1, 2.3.1.2, 4.2
1.16	24-05-2023	Daniel Joachim Larsen	Ready for review	Added section 4.9.1.3 Added section 4.1.3 Updated section 2.3.1.3
1.17	16-06-2023	Daniel Joachim Larsen	Ready for review	Added section 3.3
1.18	22-06-2023	Daniel Joachim Larsen	Ready for review	Updated WCAG description for Iframe widgets in section 2.5 Refer to KOMBIT instead of Widget Board in section 3

				Set Develop a new Widget as section 3.2.
1.19	13-06-2023	Andreas Laursen	Approved	Godkendt til R2.7

References

Reference	Title	Author	Version
[DesignGuide]	D0100 - User-Interface Guidelines - Widgets	Jesper Lund Andersen	Latest
[WidgetLocalDevelopment]	T0150 – Widget Local Development	Daniel Joachim Larsen	Latest

Table of contents

1 Background

To make Aula successful as the communication platform for all parties involved in schools and day care 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 and submit 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
- Widgets for file sharing solutions (Google and Microsoft)
- Single-sign-on links that can be placed as a toolbox on relevant Aula Dashboards

KOMBIT determines which Widget Suppliers are allowed to access the Widget Test Environment and are also responsible for approving the developed Widgets before they are put into the production environment.

1.1 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 high-level processes involved in creating new Widgets for Aula, which may be relevant for planning a Widget implementation project.

The guide is supplemented by the [DesignGuide] that explains how to replicate Aula's look and feel in the Widgets.

2 Dashboards, Modules and Widgets

Aula will be accessible via both web browser and mobile Apps.

End-users will have access to one or more configurable Dashboards that can contain both

- *Modules*: Functionality offered by the Aula itself
- *Widgets:* User interfaces from other systems that the individual municipalities and institutions use. Widgets can be either
    - *Secure Widget*: Able to personalize content according to the currently logged on user
    - *Normal Widget:* Only personalize according to basic, non-personal context data
    - A special, simple type of Secure Widget is used for Single-Sign-On links to other systems (SSO *Shortcuts*)
    - *Noticeboard Widget:* Able to personalize according to institutional data.
- *Iframe widgets:* An alternative way to build the above mentioned 4 different widgets with less restrictions on which buildtools the developers has to use. The iframe widget will function like the Secure or Noticeboard Widget, as the iframe will always have access to data from Aula.

Both Modules and Widgets are necessary for users to get the information and features they need in one place.

A Dashboard is a collection of Modules (Messages, Posts, Calendar, etc.), Widgets and other layouts (such as embedded images, logos, etc.) that are presented to a User after Login. From here, the user can navigate further via Widgets or use the Solution Modules to write Messages, create Events in Calendar etc.

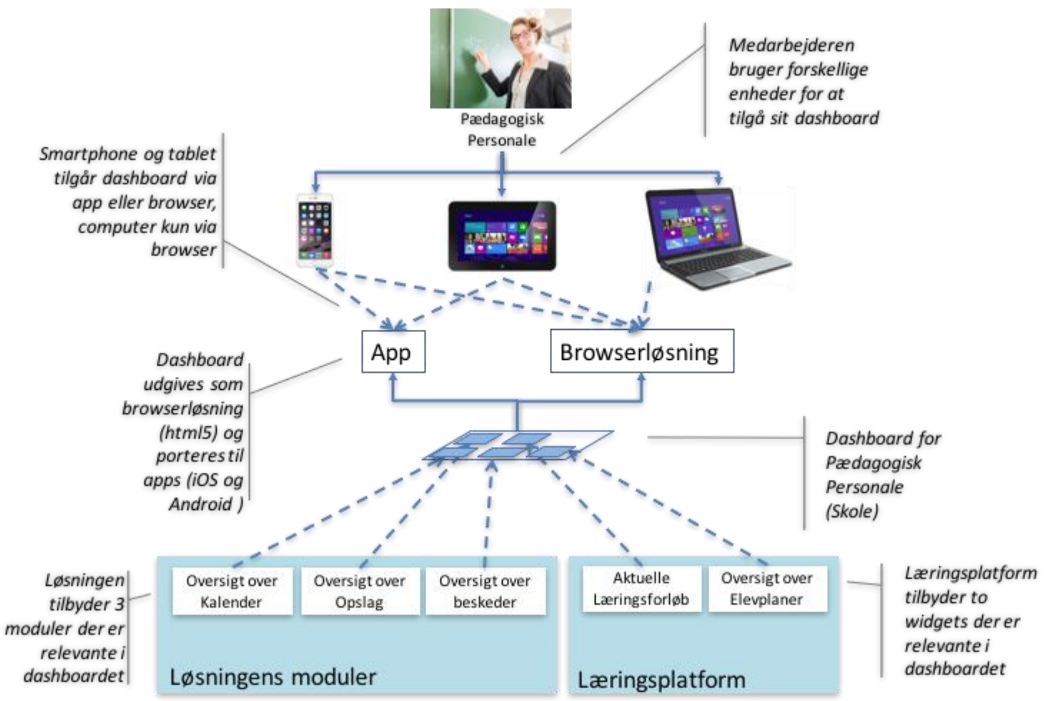

Medarbejderen bruger forskellige enheder for at tilgå sit dashboard
Pædagogisk Personale
Smartphone og tablet tilgår dashboard via app eller browser, computer kun via browser
App
Browserløsning
Dashboard udgives som browserløsning (html5) og porteres til apps (iOS og Android)
Dashboard for Pædagogisk Personale (Skole)
Løsningen tilbyder 3 moduler der er relevante i dashboardet
Oversigt over Kalender
Oversigt over Opslag
Oversigt over beskeder
Løsningens moduler
Aktuelle Læringsforløb
Oversigt over Elevplaner
Læringsplatform
Læringsplatform tilbyder to widgets der er relevante i dashboardet

Figure 1: Illustration of how Widgets are displayed alongside modules on Dashboards

2.1 User Profile Dashboards

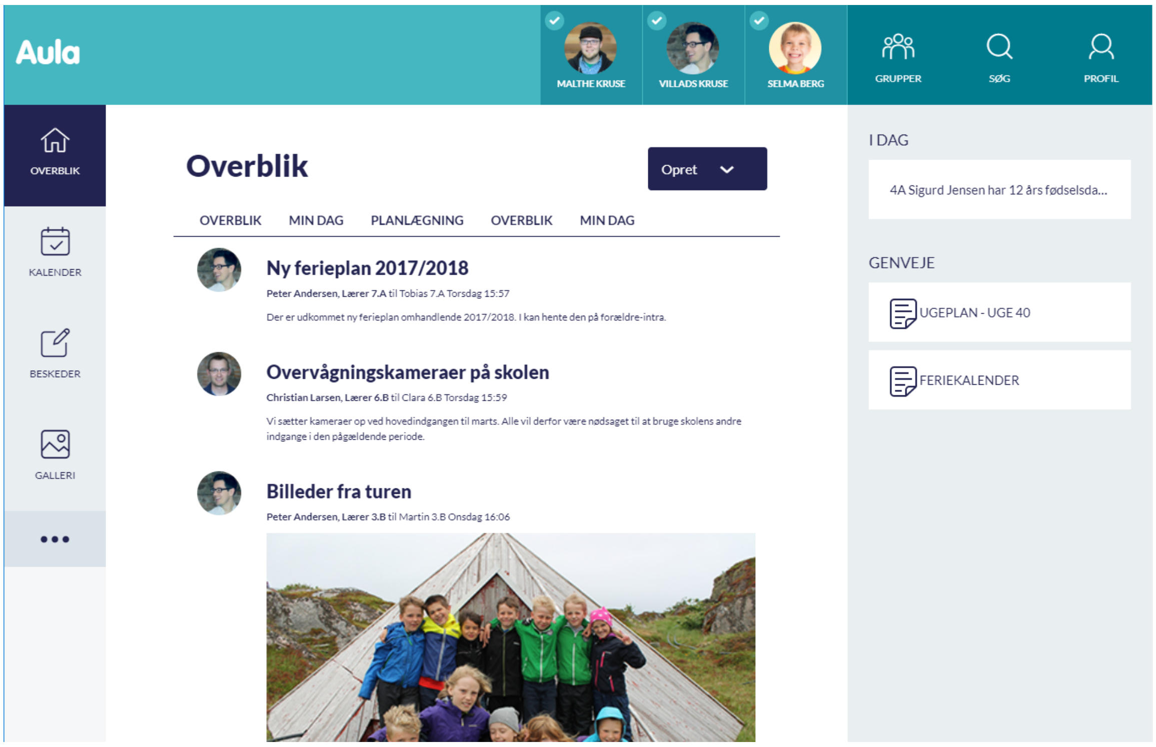
Aula will contain *User Profile Dashboards* tailored to the individual types of users:

- child
    - With separate Dashboard configurations for Daycare children and early/middle/late students

- guardian
  - Aggregating information across the institutions their children attend
- otp
  - Aggregating information across active children and active group homes.
- employee
  - With separate Dashboard configurations for Daycare employees, Teachers and Administrative staff
  - Aggregating information if the employee works at several institutions

Note, that the same Aula user can potentially be a Parent, Otp and an Employee – in this case they will have one, unique User Id (from UNI-Login), but two different profiles in Aula. At any point, the user will only have one "Active" profile in Aula, and this context information will also be available for Widgets (see section 4.2 for details about parameters).

Below is an example of such a Dashboard, in this case a "Parent" Dashboard with 4 available modules (shown in the left menu) and the first Module (Overview) currently activated.



Overblik

Ny ferieplan 2017/2018
Peter Andersen, Lærer 7.A til Tobias 7.A Torsdag 15:57
Der er udkommet ny ferieplan omhandlende 2017/2018. I kan hente den på forældre-intra.

Overvågningskameraer på skolen
Christian Larsen, Lærer 6.B til Clara 6.B Torsdag 15:59
Vi sætter kameraer op ved hovedindgangen til marts. Alle vil derfor være nødsaget til at bruge skolens andre indgange i den pågældende periode.

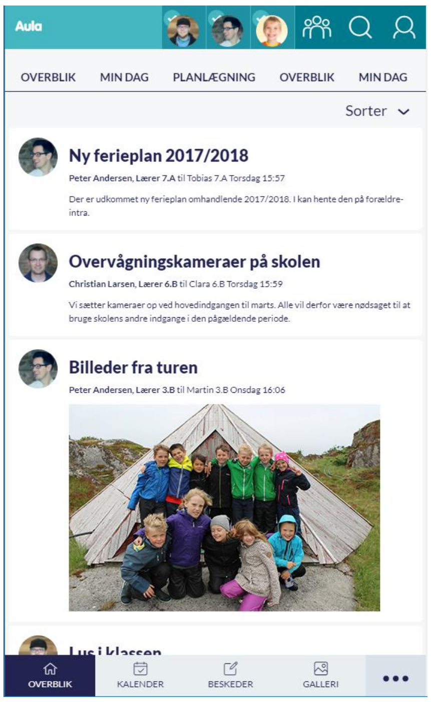
Billeder fra turen
Peter Andersen, Lærer 3.B til Martin 3.B Onsdag 16:06

4A Sigurd Jensen har 12 års fødselsda...

Figure 2: Example of a "Parent" Dashboard

Notice, that the Dashboard aggregates information across all the institutions that the parent's children attend. Aula will provide filtering for both parents (filtering per child) and teachers (filtering per institution if they work at several institutions).

All Dashboards are responsive and adjust to e.g. mobile browsers as well as the Aula Mobile App.



Ny ferieplan 2017/2018

Peter Andersen, Lærer 7.A til Tobias 7.A Torsdag 15:57

Der er udkommet ny ferieplan omhandlende 2017/2018. I kan hente den på forældre-intra.

Overvågningskameraer på skolen

Christian Larsen, Lærer 6.B til Clara 6.B Torsdag 15:59

Vi sætter kameraer op ved hovedindgangen til marts. Alle vil derfor være nødsaget til at bruge skolens andre indgange i den pågældende periode.

Billeder fra turen

Peter Andersen, Lærer 3.B til Martin 3.B Onsdag 16:06

Figure 3: Parent Dashboard when shown on Mobile browser

User Profile Dashboards are configured by

- Central Aula Administrator: Configures the typical setup of each Dashboard
- Municipality Administrator: Configures preferred dashboard configurations for the municipality
- Institution Administrators: Can also add/remove Modules and Widgets to tailor the Dashboards specifically to their preferences and needs.

Both Central Aula Administrators and Municipality Administrators can “lock” Modules and Widgets if they are not allowed to be added/removed at the next level.

Aula automatically takes care of “merging” Dashboard configurations in cases where e.g. a parent has access to two institutions with different Dashboard configurations or even from different municipalities.

As a Widget Supplier you can choose if Widget Administrators should be able to put your Widget on all the User Profile Dashboard types or if a given Widget is only relevant on some of them (e.g. a Widget that is only relevant for employees).

2.2 Group Dashboards

In addition to the User Profile Dashboards, Aula will contain *Group Dashboards* created for specific groups such as Student Project Groups, Teacher Collaboration Groups or Department Groups in day-care institutions.

These Dashboards will show up as overlays, as illustrated below. Here, the configured Modules/Widgets show up as a top-menu instead of a left-menu.

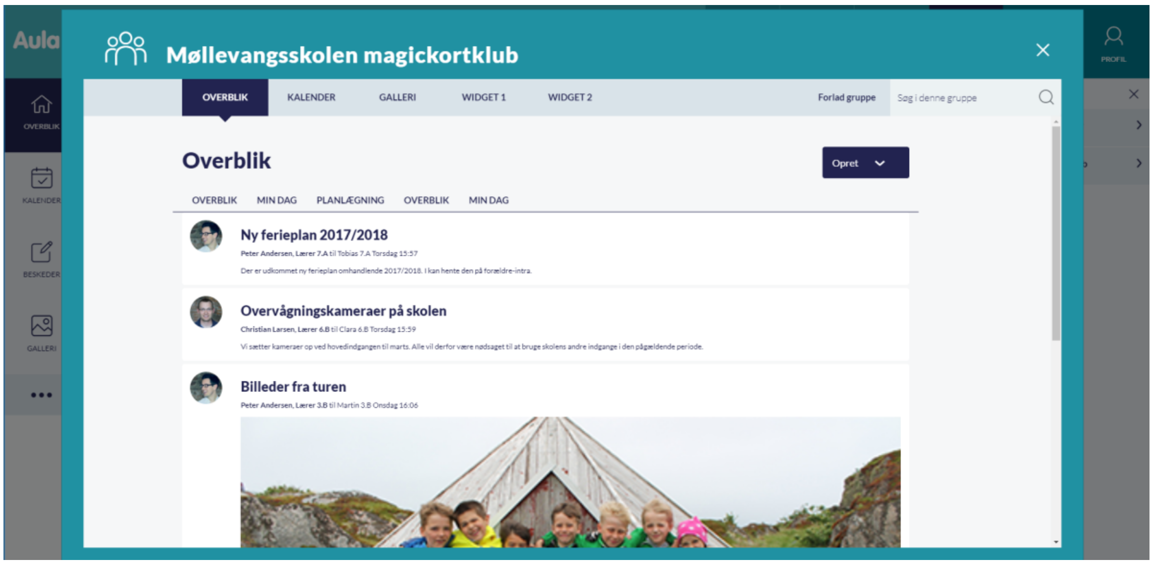


Figure 4: Example of a Group Dashboard with the first Module (Overview) active

Group Dashboards are setup specifically by the individual Group Administrator, and these do not involve any merging, aggregation or “locked” configurations.

Like the User Profile Dashboards, the Group Dashboards are also responsive and adjust to mobile browsers as well as the Aula Mobile App. Here, the configured Modules/Widgets are collapsed in a “Hamburger” menu.

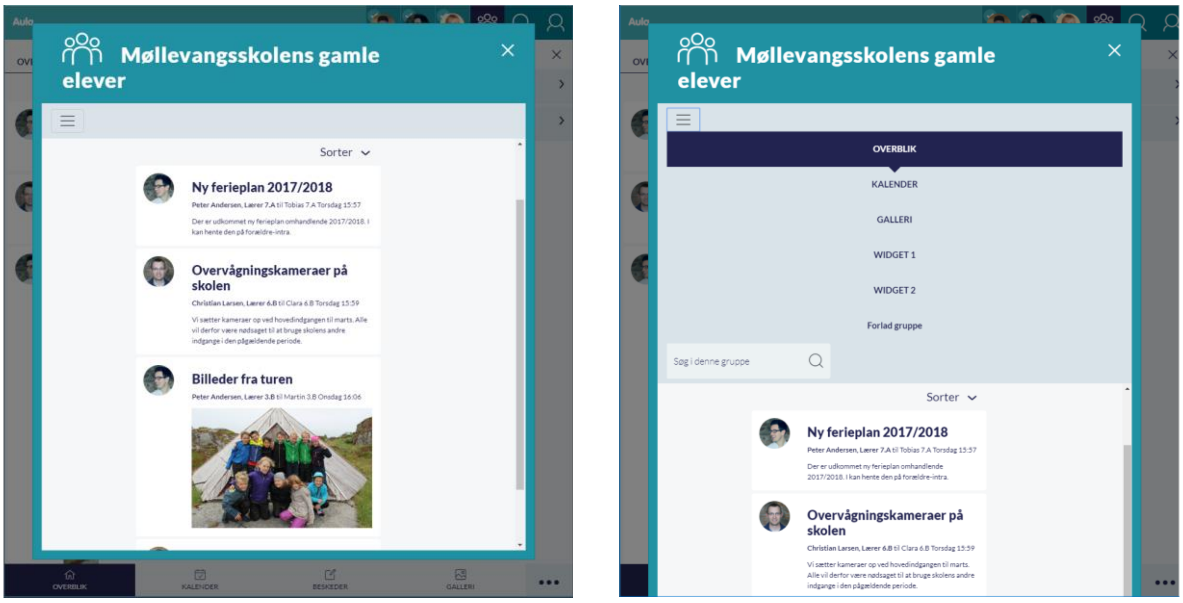


Figure 5: Group Dashboard when shown on Mobile browser (with menu expanded on the right)

As a Widget Supplier you can choose if Widget Administrators should be able to put your Widget on Group Dashboards or not.

2.3 Widgets

Widgets in Aula are developed as a small bit of combined HTML, JavaScript and CSS from a Widget Supplier. Such a Widget component runs inside the Aula Single-Page-Application (SPA), and can communicate with the Widget Suppliers systems via web services. See Section 4 for technical details about the SPA technology, Vue.js, and the communication with backend web services.

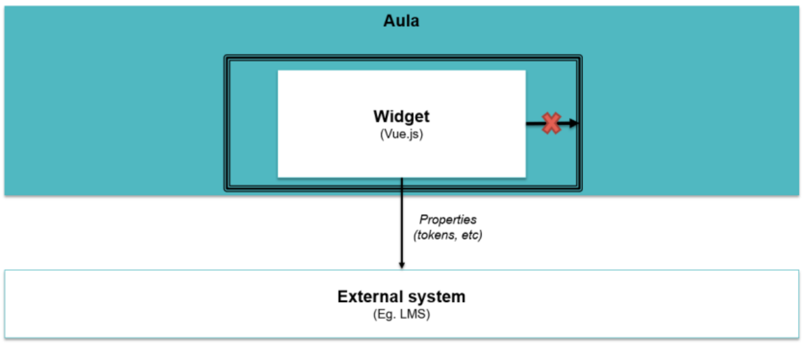


Figure 6: Widgets run inside the Aula SPA and communicate with external web services

All backend Web services (and links to Widget Supplier pages) must use https, both to ensure encryption of tokens and data and to ensure that Aula and Aula users always obtain content from the correct source.

To make the best use of the space available on a given device, Widgets must be implemented using Responsive Web Design.

Widgets can be shown in different areas on Dashboards:

1. Full page version: The Widget is shown in the main area, with no right-side content area.
    - This is typically used for Calendar views that require a lot of space.
2. Medium version: The Widget is shown in the main area, with room for a right-side content area.
    - This is the typical area for views that are central to a User Profile Dashboard
3. Narrow version: For Widgets shown in the right-side content area.
    - This is the smallest area a Widget can be put in, but also corresponds to the mobile size of a Widget and it is therefore obligatory that all Widgets can scale down to this mode.
4. Notice board: For Widgets displayed on notice boards
    - Notice boards can be configured in Aula to show multiple components, so the widget could be shown in different sizes based on the configuration of the notice board.

Full page version is illustrated below:

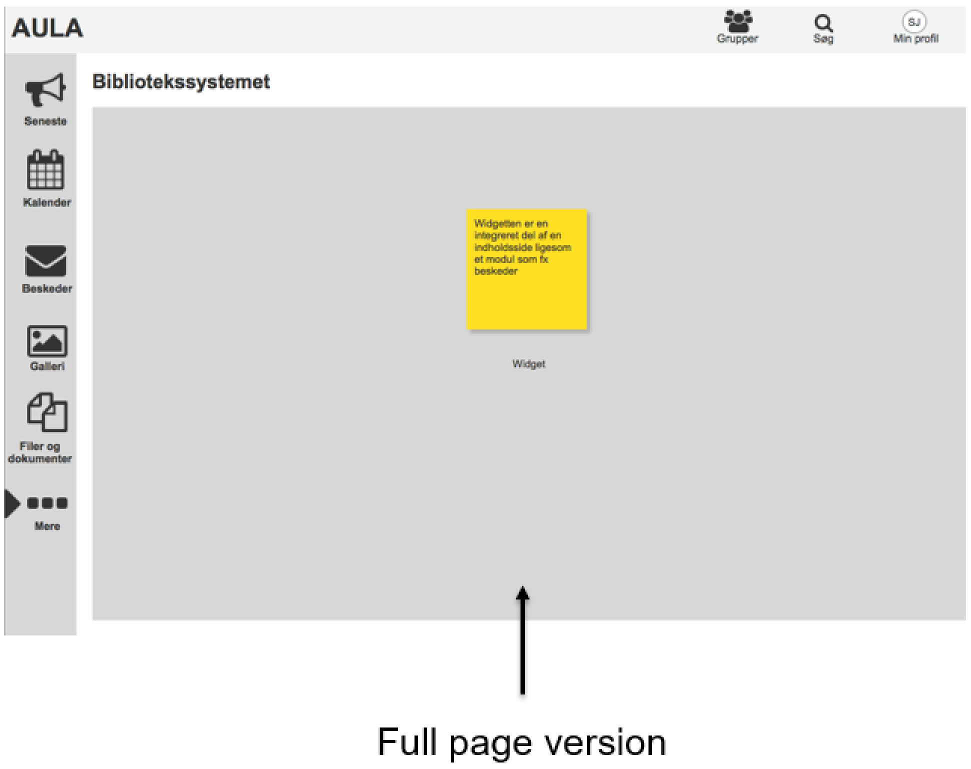


Figure 7: Widget inserted as “Full page version”

Medium/Narrow areas are illustrated below – here the UI Widgets are connected to an Aula Module on the same Dashboard, and will be displayed underneath the Module content in the Medium/Narrow areas.

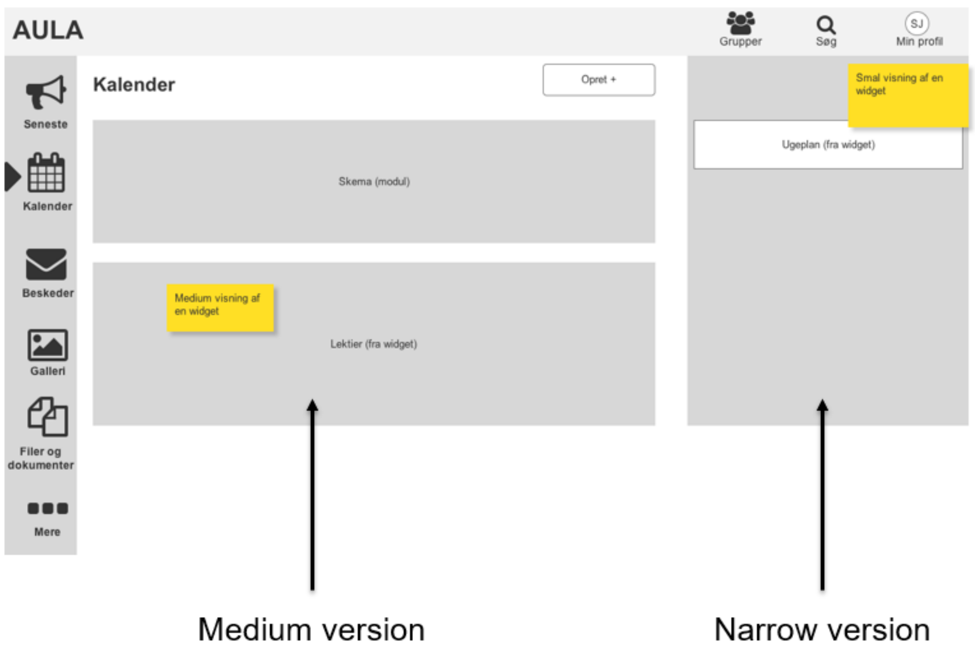


Figure 8: Widgets inserted as “Medium version” or “Narrow version” inside the Aula “Calendar” Module.

If a specific Widget has very little information to show, it is possible for the Widget Supplier to specify that the Widget does not support Full page and Medium version.

A responsive, Narrow version must always be supported in order to render any UI Widget in a Mobile mode.

See [DesignGuide] section 3 (Form factors) for further details and examples about areas, sizes and breakpoints in the Responsive Web Design of Aula.

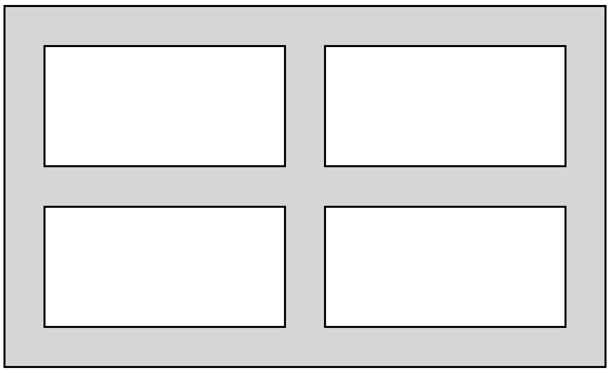
When an administrator configures a Widget on a given Dashboard, it can be placed in different areas depending on whether the Widget supports Full Page and Medium versions:

- Support for Full Page/Medium Version:
  - Can be placed as a separate, individual menu-item beneath the Aula Modules on a User Profile or Group Dashboard. Will be in Full Page version on a User Profile Dashboard and Medium version on a Group Dashboard.
  - Can also be placed as Medium version on an existing Module on a User Profile and will be rendered underneath this module in the main area (but will not have its own menu item).
- Narrow Version (which all Widgets must support):
  - Can be placed on an existing Module on a User Profile to be rendered in the right-side, Narrow version area
  - Can also be placed on an existing Module on a Group Dashboard, where the Group Administrator can place Narrow versions as he chooses in the main area.

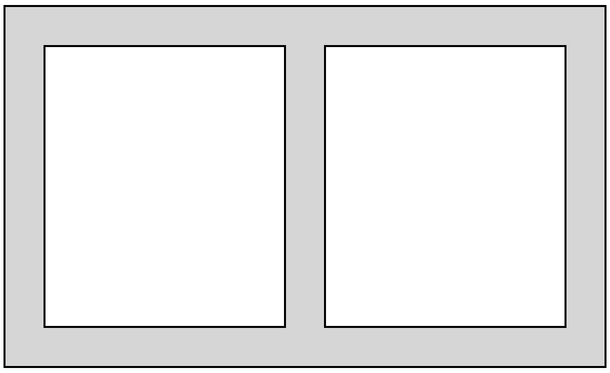
Notice board placement is different from the portal, since it resides outside the Aula portal that users are usually experiencing.

Notice boards can be set up in a variety of ways, based on how many components the administrator wish to display on the noticeboard. It is possible to use 3 different views on the test environment

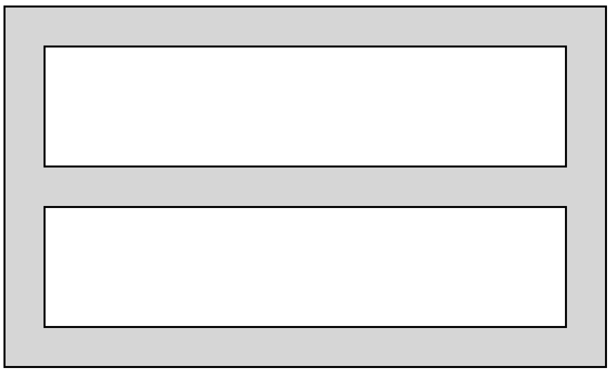
- 4x4: 4 smaller components



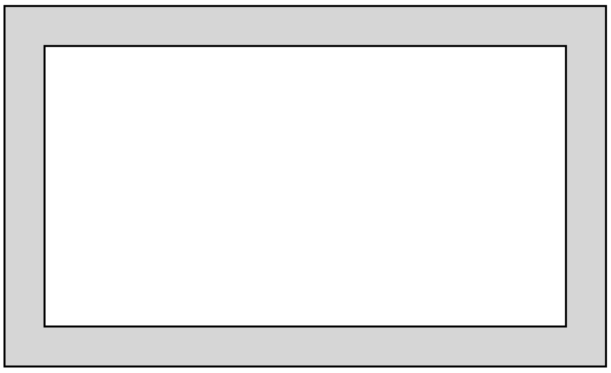
- 2xhigh: 2 components with full height side by side



- 2xwide: 2 components with full width on top of each other



- 1x1: 1 component which resembles a fullpage setting



The different views can be triggered by using the top right dropdown on the “Notice board” view in the test environment.

2.3.1.1 Normal, Secure, SSO Shortcut, Notice board and iframe Widgets

Aula distinguishes between the following types of Widgets:

- Normal: Can only personalize according to basic context data like whether the current user is child, parent, otp or employee
  - Will not have access to any information about the current user (except whether it is a child, parent, otp or employee)
- Secure: Can personalize content according to the currently logged on user
  - Will get a signed Aula Token that can be used to authenticate towards the Widget Suppliers own web services
- SSO Shortcuts: A special type of Secure Widgets that provide Widget Suppliers with the possibility to link directly from Aula into the Widget Suppliers own web applications
  - Described in further detail in section 2.4.
- Noticeboard: A widget for notice boards that is allowed to request an aula token.
  - Will get a signed Aula Token that can be used to authenticate towards the Widget Suppliers own web services. Beware that the aula token for Secure widgets differs from the aula token for Notice boards.
- Iframe: An alternative to Secure widgets or Noticeboard widget, but the widget can also act like a normal or SSO widget. It provides Widget suppliers with flexibility to use other technologies for development than is allowed for other widgets.
  - Will get a signed Aula Token that can be used to authenticate towards the Widget Suppliers own web services

  - Will have access to information about the current user if the widget is placed in users dashboards.
  - Will have access to information about the institution if placed in a noticeboard.

When Widget Suppliers develop a new Widget, they can choose between the widget types, but an already deployed Widget cannot be changed (instead you have to introduce a new Widget of the desired type).

2.3.1.2 Widget parameters

When loading a Widget, Aula will pass on some Widget parameters that can be used when calling the Widget Suppliers web services.

Widgets will always receive

- Current Session UUID: Used as a correlation ID in case there is a need to investigate system or user behavior across Aula and external systems.
- Nonpersonal context information like which Dashboard area the Widget is placed in, whether the Dashboard is displayed in a browser or a Mobile App and whether it is shown for a specific week number (if users can page through calendars on the Dashboard)

Secure Widgets, SSO Shortcuts and iframe receive

- A signed Widget token from Aula with basic information about the UNI-Login user who is logged in. This can be used for authenticating with the Widget Suppliers web services.
- Relevant context information for personalization of Widget content (e.g. which of the parent's children the Dashboard is showing, whether the user is signed in with 2-factor authentication).

Notice board widgets and iframe widgets placed in a notice board receives

- A signed Widget token from Aula with the institution code and notice board id for the notice board the widget is placed on. This can be used to verify the request to ensure it originates from a notice board.

For Secure and iframe Widgets, users will always be identified with their UNI-Login ID. If the Widget requires more detailed information about users (names, emails, addresses) and relations between them (e.g., parents and children), the Widget Supplier will have to obtain such information directly from UNI-Login.

The specific parameters available are listed in Section 4.2.

2.3.1.3 Widget notifications

Aula will use small notification numbers to alert the user to new, unread content.

Currently there are 3 places that badges for widgets are shown:

First is badges on the User Profile Dashboards; this badge notification is shown as a small number in the left-side menu, indicating that this Module or Widget has unread content.

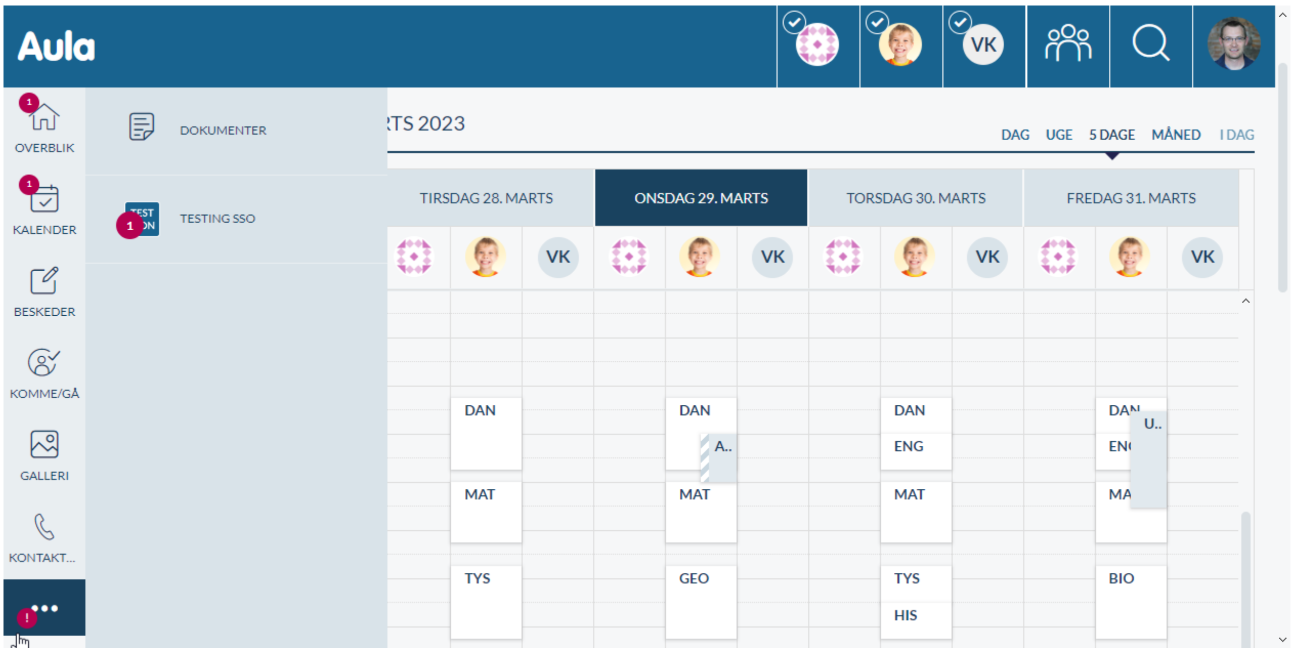
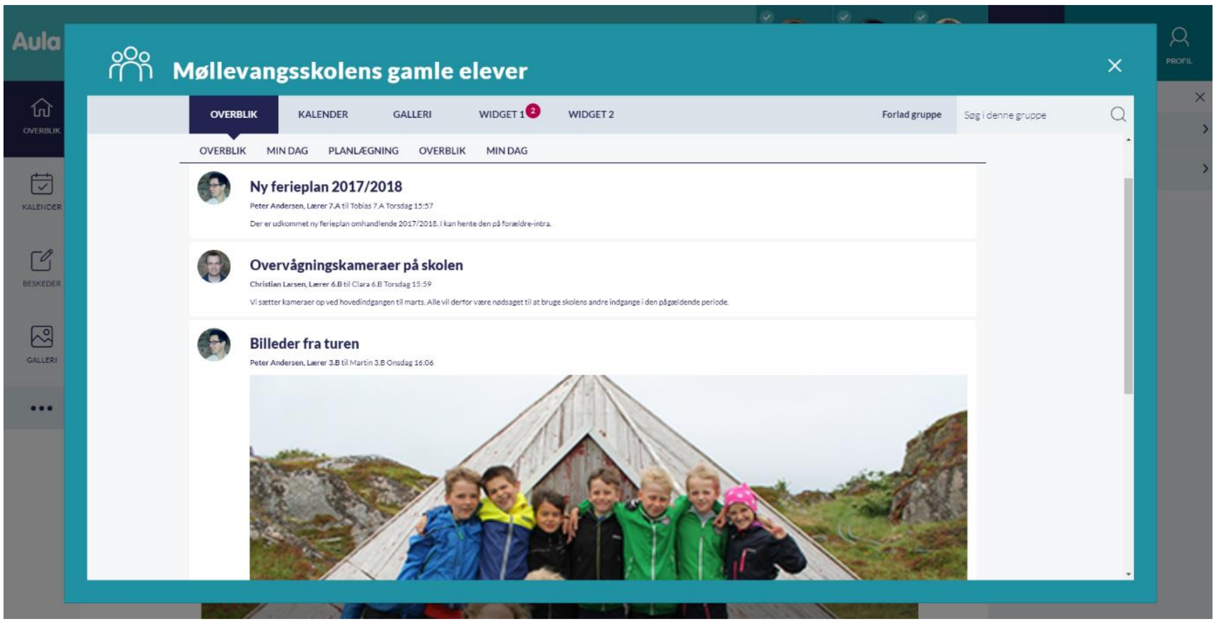


Figure 9: Notification on a Secure Widget placed in the User Profile Dashboard

Secondly, on the top-level menu when you are inside a Group Dashboard, or on the top-level menu when using the phone, when widgets are placed in the Calendar ("Kalender") or in Posts (Overblik) module.



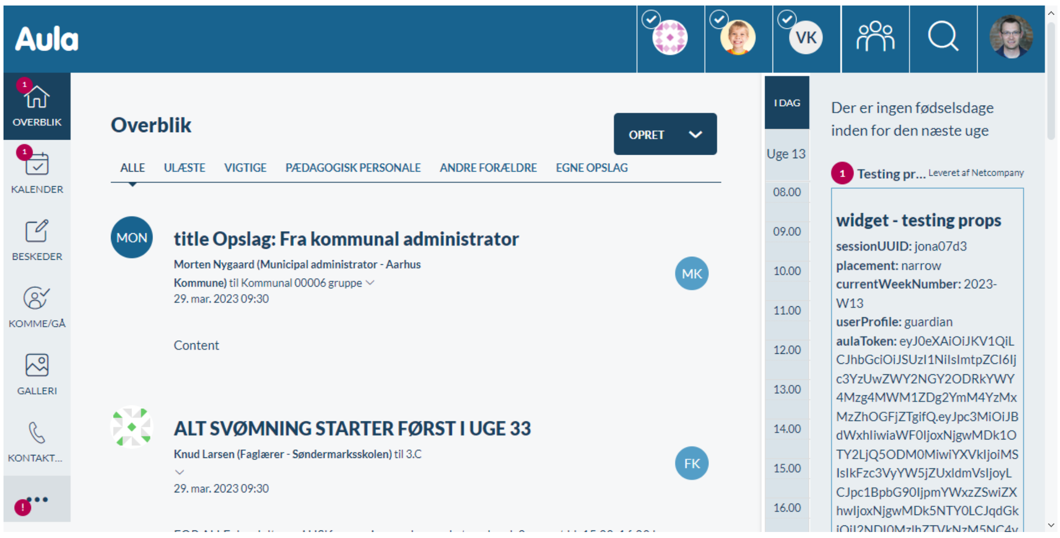
Ny ferieplan 2017/2018
Peter Andersen, Lærer 7.A til Tobias 7.A Torsdag 15:57
Der er udkommet ny ferieplan omhandlende 2017/2018. I kan hente den på forældre-intra.

Overvågningskameraer på skolen
Christian Larsen, Lærer 6.B til Clara 6.B Torsdag 15:59
Vi sætter kameraer op ved hovedindgangen til marts. Alle vil derfor være nødsaget til at bruge skolens andre indgange i den pågældende periode.

Billeder fra turen
Peter Andersen, Lærer 3.B til Martin 3.B Onsdag 16:06

Figure 10: Notifications on Group Dashboards

The third place Widget badges are shown, is when a widget is added in the side menu on Posts (Overblik), or below the Calendar ("Kalender").



title Opslag: Fra kommunal administrator
Morten Nygaard (Municipal administrator - Aarhus Kommune) til Kommunal 00006 gruppe
29. mar. 2023 09:30

Content

ALT SVØMNING STARTER FØRST I UGE 33
Knud Larsen (Faglærer - Søndermarksskolen) til 3.C
29. mar. 2023 09:30

Der er ingen fødselsdage inden for den næste uge

1 Testing pr... Leveret af Netcompany

widget - testing props
sessionUUID: jona07d3
placement: narrow
currentWeekNumber: 2023-W13
userProfile: guardian
aulaToken: eyJ0eXAiOiJKV1QiLCJhbGciOiJSUzI1NiIsImtpZCI6Ijc3YzUwZWY2NGY2ODRkYWY4Mzg4MWM1ZDg2YmM4YzMxMzZhOGFjZTgifQ.eyJpc3MiOiJBdWxhIiwiaWF0IjoxNjgwMDk1OTY2LjQ5ODM0MiwiYXVkIjoiMSIsIkFzc3VyYW5jZUxldmVsIjoyLCJpc1BpbG90IjpmYWxzZSwiZXhwIjoxNjgwMDk5NTY0LCJqdGkiOiI2NDI0MzlhZTVkNzM5NC4y

Figure 11 Notifications on Posts (Overblik) side panel

Widget Suppliers can send a notification through the external widget notifications API, see section 4.5 for a technical description of how to send and handle notifications. The Widget Suppliers are responsible for removing the badge notifications for the user one notification at a time, so that they don't stay after the content has been read.

A Widget Supplier can request Aula to send out a push notifications through an API that is provided by Aula, Aula's notification handler will send a notification to the users' phone or e-mail address. Push notifications send through the API provided by Aula are send and forget, it is not possible to know if they are received and they cannot be altered or deleted. A user in Aula can decide how and if they receive push notifications:

- Instantly sent push notifications on the phone
- Instantly sent emails to the users registered email address
- Delayed email push notifications where notifications will be bundled
- No notifications from the widget

The push notification for phones will have the following text "New activity from [Widget Name]" (The actual message will be Danish: "Ny aktivitet fra [Widget name]")

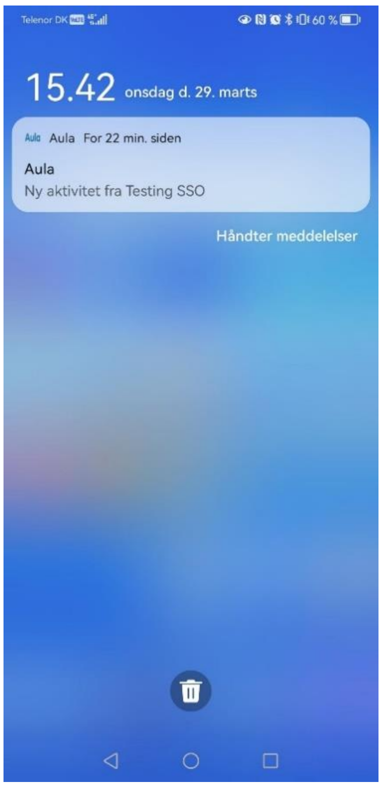
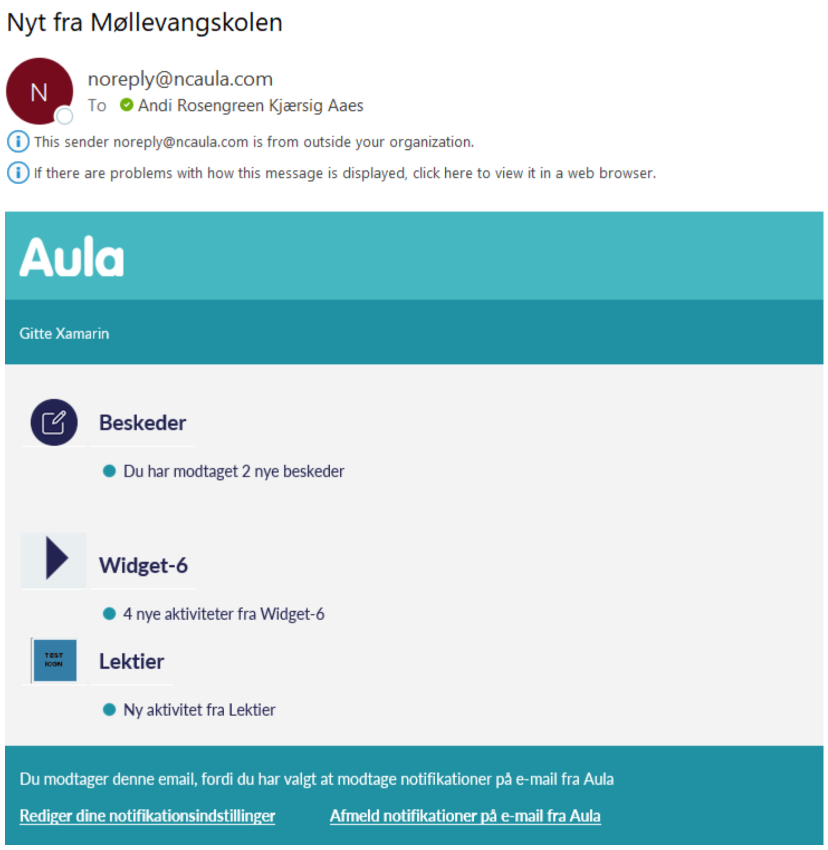


Figure 12: A push notification sent to the users phone, from the widget "Testing SSO"

When a push notification is sent to the user an e-mail will be generated with the name of the widget, the email will include all open notifications present at that time. Note that the user may not receive the notification immediately after it is sent. In Aula, users have the option to receive push notifications either via email or on their phone.



Nyt fra Møllevangskolen

noreply@ncaula.com
To Andi Rosengreen Kjærsig Aaes

This sender noreply@ncaula.com is from outside your organization.

If there are problems with how this message is displayed, click here to view it in a web browser.

Aula

Gitte Xamarin

Beskeder

- Du har modtaget 2 nye beskeder

Widget-6

- 4 nye aktiviteter fra Widget-6

Lektier

- Ny aktivitet fra Lektier

Du modtager denne email, fordi du har valgt at modtage notifikationer på e-mail fra Aula

Rediger dine notifikationsindstillinger   Afmeld notifikationer på e-mail fra Aula

Figure 13: A push notification sent to the users e-mail, from the widgets "Widget-6" and “Lektier”

See Section 4.5 for the technical details about implementing notifications in Widgets.

2.4 SSO Shortcuts

Single-Sign-On Shortcuts are a special type of Secure Widgets that provide Widget Suppliers with the possibility to link directly from Aula into the Widget Suppliers own web applications, with a sign-on token containing the Aula user’s UNI-Login Id (essentially providing Single-Sign-On for the Aula user).

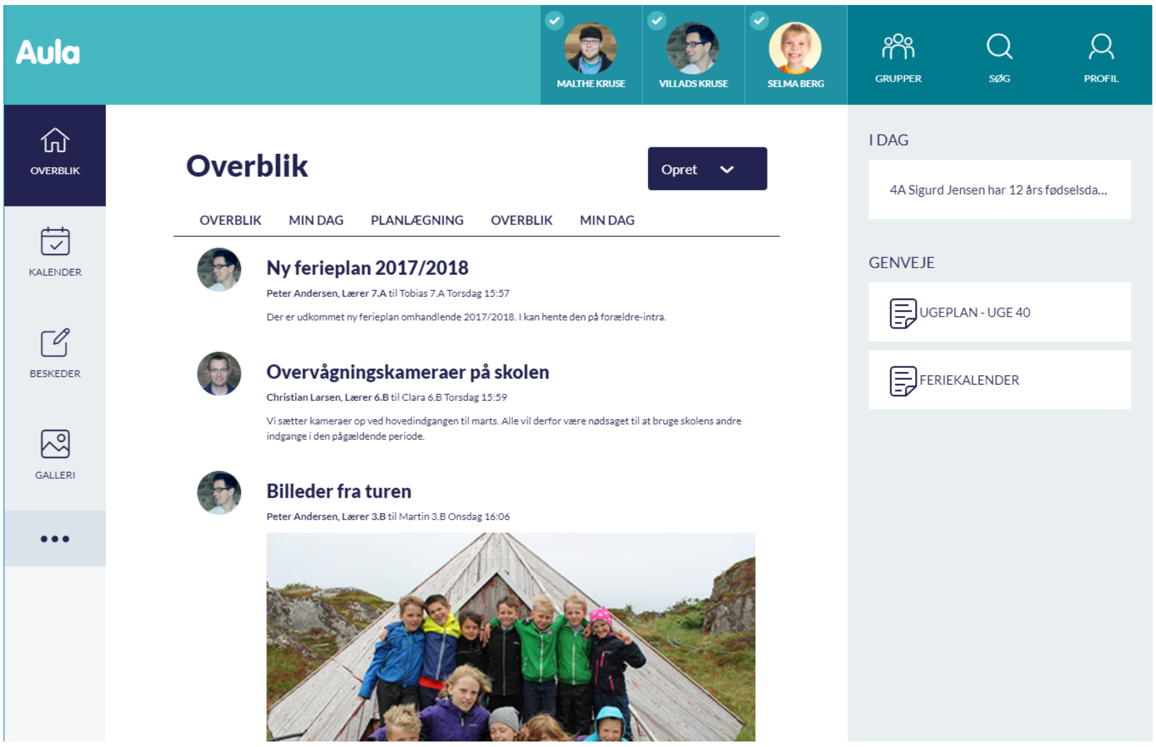
Such web links can also be made inside the HTML template of ordinary Secure Widgets (see Section 4.6 for details), but the concept of SSO Shortcuts makes it easy for Widget Suppliers to provide such shortcuts, without having to write a complete Widget.

SSO Shortcuts are treated and approved like any other Secure Widget, but a Widget Supplier only needs to provide

1. URL (must be a https URL as encryption of the Aula sign-on token is required)
2. Title
3. Description
4. An optional icon (See [DesignGuide] section 6.3 for examples and guidelines for icons in Aula).

The SSO Shortcut can then be placed by Dashboard Administrators on Modules inside the various Aula Dashboards. Like with any other Widget, the Widget Supplier can choose whether the SSO Shortcut can be used on all or only specific User Profile Dashboards, and whether the SSO Shortcut can be placed on Group Dashboards.

As shown below, SSO Shortcuts are rendered in the right-side column, beneath any other components on the specific Module.



4A Sigurd Jensen har 12 års fødselsda...

Ny ferieplan 2017/2018
Peter Andersen, Lærer 7.A til Tobias 7.A Torsdag 15:57
Der er udkommet ny ferieplan omhandlende 2017/2018. I kan hente den på forældre-intra.

Overvågningskameraer på skolen
Christian Larsen, Lærer 6.B til Clara 6.B Torsdag 15:59
Vi sætter kameraer op ved hovedindgangen til marts. Alle vil derfor være nødsaget til at bruge skolens andre indgange i den pågældende periode.

Billeder fra turen
Peter Andersen, Lærer 3.B til Martin 3.B Onsdag 16:06

Figure 14: SSO Shortcuts shown on a Dashboard (“Genveje”)

Clicking such a SSO Shortcut will open the relevant web link in a new browser windows (or tab), essentially doing a POST of a sign-on token along with the current Aula context (e.g. whether you are logged on as parent or teacher).

Section 4.5 describes the details about how the Widget Supplier's web site should receive and validate this sign-on token from Aula.

Due to limitations in iOS webviews, SSO shortcuts will be send as GET from the app.

Due to Aula handling OTPs and guardians as two separate user types, and UNI-Login treating them as a single user type, SSO shortcuts are disabled for users logged in as OTPs, as to not create any confusion about what children they should be shown in case they are also guardians in their private life.

2.4.1 Logout

Note, that while Aula provides a sign-on token there is no single-log-out mechanism in Aula. Logging out of Aula does not automatically log you out of the SSO Shortcut browser windows and likewise, logging out of these SSO Shortcut browser windows does not log you out of Aula. This approach reflects the idea of SSO Shortcuts as something that makes it easy to open up separate systems, rather than a way of in-lining HTML from several systems in a view with the same login session.

Users that open up SSO Shortcut browser windows in this way will therefore have to remember to log out of these separate browser windows themselves (or close the browser completely).

2.5 Iframe Widgets

An iframe Widget uses the native HTML element iframe. This element is a completely separate webpage that is constrained to a smaller part of Aula, and all functionality in the iframe is handled by the Widget Supplier. This gives the Widget Supplier a lot of flexibility while developing Widgets, as the restrictions that apply to non-iframe Widgets does not apply, for example iframe Widgets are framework agnostic, so a widget supplier can develop iframe widgets that are aligned with their own organization.

The iframe widgets still have some constraints, but it is the responsibility of the Widget Supplier to comply with all restrictions, as it will not be possible for Aula to enforce.

- Must comply with the law as a solution developed by a municipality in Denmark. Some examples are listed below:
    - GDPR
        - This also includes any cookies that are not strictly necessary technical cookies
    - No advertisements
    - No propaganda
    - Web accessibility ref. https://www.retsinformation.dk/eli/lta/2018/692
- It is not allowed to use any external resources outside the widget vendors domain.
    - Exceptions can be approved by creating a case in the toolkit.
- An iframe Widget is subject for re-approval before changes are deployed to the Widget. See section 3.4 for more information about modifying an existing iframe Widget.

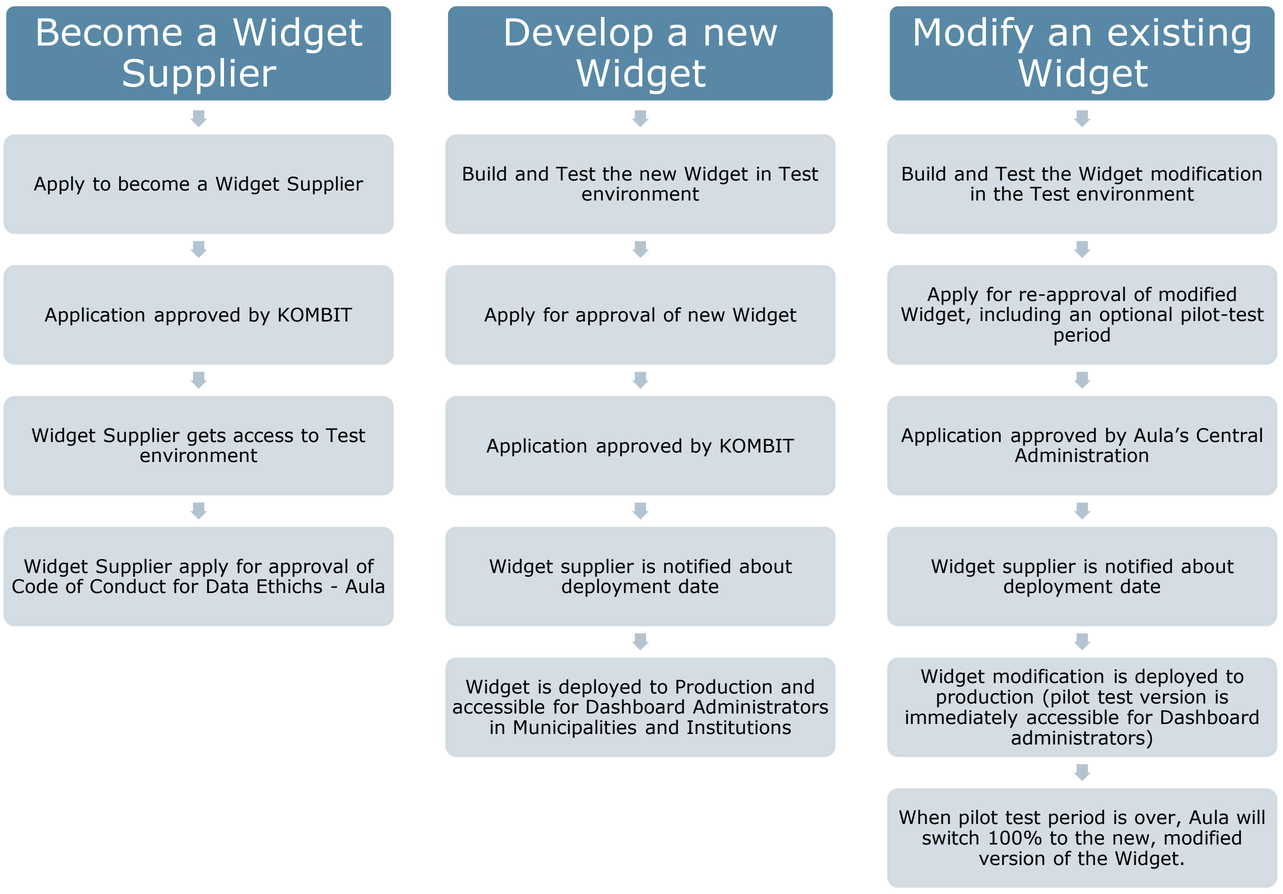
3 Widget Processes

There are three main processes involving Widget Suppliers in Aula:

1. Becoming a Widget Supplier in Aula
2. Developing a new Widget for Aula
3. Modifying an existing Widget already deployed in Aula

The processes are the same for full Widget Development and for the simpler SSO Shortcuts, although the approval of SSO Shortcuts is much faster as there is less to review.

The illustration below summarizes the steps in the processes, which are described in the sections below.


Become a Widget Supplier
Apply to become a Widget Supplier
Application approved by KOMBIT
Widget Supplier gets access to Test environment
Widget Supplier apply for approval of Code of Conduct for Data Ethichs - Aula
Develop a new Widget
Build and Test the new Widget in Test environment
Apply for approval of new Widget
Application approved by KOMBIT
Widget supplier is notified about deployment date
Widget is deployed to Production and accessible for Dashboard Administrators in Municipalities and Institutions
Modify an existing Widget
Build and Test the Widget modification in the Test environment
Apply for re-approval of modified Widget, including an optional pilot-test period
Application approved by Aula's Central Administration
Widget supplier is notified about deployment date
Widget modification is deployed to production (pilot test version is immediately accessible for Dashboard administrators)
When pilot test period is over, Aula will switch 100% to the new, modified version of the Widget.

Code of Confuct for Data Ethichs application can be submitted to mailto:aula@kombit.dk

3.1 Become a Widget Supplier

The first step of supplying Widgets for Aula is to apply to become a Widget Supplier. This is done by submitting following Ansøgning-om-adgang-til-aula-Widget-testmiljø.pdf application to aula@kombit.dk, with company information, contact information as well as reason for wanting to supply Widgets to Aula.

On a regular basis, KOMBIT will consider and approve all relevant applications.

The approved Widget Suppliers then retrieve a username/password login to their own area of the Widget Test environment, also known as the Aula External Test 2 Environment, where they can now start developing and testing their own Widgets. Additionally, there is a local widget development environment available, Refer to [WidgetLocalDevelopment], to get started on widget development locally.

3.2 Develop a new Widget

To ensure that Aula stay a secure and safe environment to share sensitive data, the review will focus on these points in regards to security for non-iframe widgets:

- No use of the v-html tag as this potentially opens up for XSS. A component for this is supplied by Netcompany, see under 5.4 HTML component.
- No code trying to read/write directly from Aula state.
- No code trying to read Aula data by traversing through the Vue tree (this.$parent, this.$parent.$parent etc.).
- Make sure that the use of media is restricted in regards to XSS.
- There may not be any references to external Javascript libraries.
- Don't use any of Aulas global Vue components unless permission to use component have been given.

The only technical requirement for iframe widgets is that there may not be any references to external resources.

These points are all considered as severe security issues and the widget will not be approved if any of these are present.

Each Widget Supplier will have their own area inside the Widget Test Environment, and both administrative access and test access is 100% browser based. Their username/password access enables Widget Suppliers to upload new Widget definitions as well as upload new versions of existing Widgets.

Note, that Aula does not store historic versions of the Widgets, but we recommend that Widget Suppliers use their own source control system to store and compare historic versions of Widget source code.

3.2.1 Building and testing the new Widget

Widget Suppliers can use their preferred editor to develop the Widget source code (see Section 4.1 for technical details).

To test a new Widget, Widget Suppliers need to log in to their dedicated area within the Widget Test environment. Alternatively, Widget Suppliers can build and test their widget in the local development environment. For instructions on setting up the development environment, refer to [WidgetLocalDevelopment]. Create the Widget by providing the necessary data (see Section 4.3 for technical details and data specifications):

- Whether it is a Normal Widget, Secure Widget, Iframe Widget or SSO Shortcut
- Name
- Description
- Icon (optional – see [DesignGuide] for examples and guidelines for icons in Aula).
- Whether it is available across all User Profile Dashboards and Group Dashboards
- For SSO Shortcuts:
    - The URL to POST, sent by GET for app, the sign-on token to.
- For Normal, Secure:
    - The actual Widget Code
    - Whether it can be used for Full page/Medium areas
- For Noticeboard
    - The actual Widget code

    - Only use the “Notice board” placement to test the widget
- For Secure:
    - An optional Notification Handler script if the Widget should show an “unread” content counter in the menu (see Section 2.3.1.3 for details)
- For Iframe Widgets:
    - An URL to point to the hosted domain of the iframe.
    - The actual Widget code
    - Whether it can be placed as a full page, medium or noticeboard.
    - An optional Notification Handler script if the Widget should show an “unread” content counter in the menu (see Section 2.3.1.3 for details).
        - Is not applicable if the iframe is placed as a noticeboard

The Aula platforms build tools then take care of processing the Widget code, and if the Widget Supplier has uploaded code that cannot be parsed, only the specific Widget Supplier area will be affected.

When setting up a new SSO Shortcut, no actual development is necessary, as it only involves specifying a URL and the same metadata as mentioned above.

To test the new Widget, the Widget Supplier has both a User Profile Dashboard and a Group Dashboard in their Widget Supplier Area of the Widget Test Environment.

Here, the Widget Supplier can test how the Widget works, and also specify exactly which context parameters to render the Dashboard with (hereby testing the various scenarios the Widget can experience in Aula).

Context parameters that can be specified are:

- Test Uni-Login ID for the current user viewing the Dashboard (used in the Aula Token issued to the Widget)
- Assurance Level (2/3)
- Institution code (only used for notice boards)
- User Profile (whether the current user is logged on as child, parent, otp or employee)
- Child Filter (UNI-Login ID of the children a parent is filtering on)
- Institution Filter (UNI-Login institution ID of the institutions the Widget should show content for)
- Group Id (UNI-Login group Id if the Widget is shown on a specific Group Dashboard)
- Placement (whether the Widget is placed in the Full Page, Medium or Narrow area inside Aula)
- Current Week Number (for paging in calendar views)
- Is Mobile App (in order to simulate that the Dashboard is rendered in the Mobile App)

The context parameters can be changed dynamically, which will send events to the Widget so that you can simulate e.g. whether changing a Child Filter or forwarding Current Week Number has the desired effect.

Using the Dashboards and the configurable properties, the Widget Supplier can test various cases such as:

- Using a signed Aula Token to communicate with their Web API and retrieving data based on the configured test UNI-Login Id
- Showing the right content based on student/parent/employee profile and filters on Children and Institutions

- Hiding sensitive information if Assertion Level is less than 3 (if the user is not logged on with two-factor authentication)
- Different placements and responsive sizes of the Widget

The Dashboards will also provide the necessary functionality to test that Widgets react on Aula events such as:

- Changing the current week/date if showing side-by-side calendars across Widgets
- Clicking an SSO Shortcut with an Aula sign-on token

3.2.2 Applying for approval of a new Widget

When a Widget Supplier has finished testing a new Widget, the resulting Widget definition is submitted for approval online in the Widget Test Environment, using the same username/password as for developing Widgets.

As part of the approval, the Widget Supplier describes

- The purpose of the Widget
- The Municipalities/Institutions that have requested the Widget (if any)
- Whether the Widget requires access to new external resources, and why (see Section 4.7 for details about this)
- Remarks to consider when reviewing the Widget (e.g. a deadline, other Widgets that are related and should be reviewed at the same time)
- Contact information (email and phone number) for review comments
- For Iframe widgets widget suppliers are required to upload the source code in a zip file, the first time the widget is deployed, for review purposes.

3.2.3 Approval by KOMBIT

On a regular basis KOMBIT will consider and approve all relevant applications, both in terms of allowing access to sensitive data and in terms of technical requirements and best practices (see Sections 4.8 and 4.9 for details).

Widget Suppliers with approved Widgets will be informed about the approval and will also be kept up-to-date about the actual deployment time.

Widget Suppliers with declined applications will be informed about the reason why the Widget cannot be approved. Depending on why it has been declined the Widget Supplier will either

- Be able to resubmit some minor changes that can be approved by the Central Administrators of Aula (currently assumed to be twice per month)
- Have to resubmit a completely new application for approval of the New Widget

3.2.4 Notification of deployment date

As mentioned, Widget Suppliers with approved Widgets will be notified about the exact deployment time.

It is up to the individual Widget Supplier to contact relevant Municipalities and Institutions and inform them about when and how to put the new Widget on their Dashboards in Aula.

3.2.5 Deployment to production

Approved Widgets will be deployed to the production environment periodically (assumed to be twice per month).

When deployed, the new Widgets will be available to Aula administrators (both at Municipality and Institutional level) who can begin assigning them to Dashboards.

In case of deployments being re-scheduled or rolled back, all relevant Widget Suppliers will be informed immediately.

For Iframe widgets, deployment is handled by the individual widget suppliers, following the initial approval.

3.3 Modifying an existing Widget

Updating existing Widgets follows the same process as developing a new process, except that

- Approval can often be done by Aula Central Administration rather than KOMBIT
- When deploying a modification to an existing Widget the Widget Supplier can choose a pilot period before all users and institutions switch to the new version

3.3.1 Building and testing the modifications

Just like when building new Widgets, Widget Suppliers can use their own area of the Aula Widget Test Environment freely to update existing Widgets or run several Widget versions side-by-side.

Again, we would like to point out that Aula does not store historic versions of the Widgets and that we recommend that Widget Suppliers use their own source control system to store and compare historic versions of Widget source code.

3.3.2 Applying for re-approval of a modified Widget

The main difference from developing a new Widget is, that when you submit a change to an existing Widget you need to submit a re-approval request instead of a normal approval request:

- In re-approval you cannot change whether a Widget is Secure or Normal.
- In re-approval, Widget Suppliers can optionally ask for a Pilot Period, allowing the old version to run alongside the modified one until a specific date.

As part of the approval, the Widget Supplier describes

- The reason for changing the Widget
- Remarks to consider when reviewing the Widget (e.g., a deadline, other Widgets that are related and should be reviewed at the same time)
- Contact information (email and phone number) for review comments

Again, this is done online in the Aula Widget Test Environment using the same username/password as for developing Widgets.

3.3.3 Re-approval by Aula’s Central Administration

As most modifications only involve simple source code changes to the existing Widget, the review will normally not involve KOMBIT.

On a regular basis, the Aula Central Administration will consider and re-approve all such source code changes in terms of technical requirements and best practices (see Sections 4.8 and 4.9 for details).

Widget Suppliers with approved Widget Modifications will be informed about the approval and will also be kept up-to-date about the actual deployment time.

Widget Suppliers with declined applications will be informed about the reason why the Widget cannot be re-approved. Depending on why it has been declined the Widget Supplier will either

- Be able to resubmit some minor changes that can be re-approved by the Central Administrators of Aula
- Have to resubmit a completely new application for approval of the Widget

3.3.4 Notification of deployment date

Widget Suppliers with re-approved Widgets will be notified about the exact deployment time.

It is up to the individual Widget Supplier to contact relevant Municipalities and Institutions and inform them about how and when they will see the modifications in Aula.

If the Widget Supplier has chosen to use a Pilot Period, the Widget Supplier should inform the relevant Municipalities and/or Institutions that they will be able to switch to this pilot version when the modifications are deployed to production.

3.3.5 Deployment to production

Re-approved Widgets will be deployed to the production environment periodically (assumed to be twice per month).

If a Pilot Period has been specified, there will be no immediate change to the Widget in Aula. Instead, the pilot test version will be made accessible for all Dashboard Administrators who can manually select to use the new pilot version instead of the stable version of the Widget.

If no Pilot Period has been specified, the new Widget version automatically replaces the old one across all Aula Dashboards.

In case of deployments being re-scheduled or rolled back, all relevant Widget Suppliers will be informed immediately.

3.3.6 Pilot test period is over

When the specified Pilot Period is over, the new Widget version automatically replaces the old one across all Aula Dashboards.

Should the Widget Supplier regret the Widget Modifications during the Pilot Period, the Aula Central Administration will be able to delay or cancel the deployment of the Widget Modifications.

3.4 Modifying an existing iframe Widget

This section explains the technical process of re-approving Iframe Widgets.

Generally Iframe widgets do not require reviews for each update as long as they are considered minor updates (See section 3.4.2 for further information), however Aula reserves the right to request the widget supplier to provide their widget code, for review purposes. This ensures transparency and allows Aula to maintain security standards.

3.4.1 Building and testing the modifications

Just like when building new iframe Widgets, Widget suppliers can use the local widget development environment Ref. [WidgetLocalDevelopment] freely to update existing Iframe Widgets.

3.4.2 Re-approving an iframe widget

Aula can't prevent a larger change from occurring to an iframe widget, but Aula does require that widget suppliers submit their iframe widget code base for a review prior to making a larger update to their iframe widget.

A larger update to a widget iframe is considered a change to the existing logical operations of the iframe widget, fx, implementing or removing a feature in the iframe widget or an update to the existing functionality is also considered a larger update.

Cosmetic changes as well as smaller bug fixes are not considered a larger update and are not subject to a re-review.

3.4.3 Notification of deployment date

With iframe widgets, the Widget supplier is in full control of their own deployment schedule.

Widget suppliers are expected to submit their iframe widget code to Aulas toolkit and secure approval for larger updates prior to implementing any changes to the iframe widget in production.

Should it be discovered that a major update to an iframe widget has been implemented without prior approval, Aula reserves the right to disable the supplier's widget in production until the update has been formally approved.

3.5 Other processes

Two other, smaller processes regarding Widgets in Aula are worth mentioning, namely disabling Widgets temporarily and removing a Widget completely from Aula.

3.5.1 Disabling a Widget temporarily

Aula will have the ability to quickly disable a specific Widget and either hiding it completely or replacing it with a relevant error message.

The reason for wanting to disable a Widget can for instance be sudden security/capacity/performance risks or sudden unforeseen issues that a Widget Supplier is facing with their Widget code or backend Web services.

This process can be initiated both by Widget Suppliers, Aula Central Administration and KOMBIT.

Depending on the reason for disabling the Widget, enabling it again may require formal approval, but only in cases with security risks or serious performance/capacity issues.

3.5.2 Removing a Widget from Aula

Widget Suppliers can ask to have a Widget removed completely from Aula, but in order to do this they first have to inform the relevant Municipalities and/or institutions about the impending removal.

The Widgets will be removed as part of the regular Widget Deployment routine, currently assumed to be twice per month.

4 Technical guide

This section describes the technical details for developing Widgets for Aula. Refer to [DesignGuide] for guidance related to development of the user-interface related parts of a Widget.

4.1 Technology

The Aula front-end uses the Vue.js 2.x framework, a progressive JavaScript framework for building user interfaces. The framework is in the same category as frameworks such as React, Angular, Backbone etc.

4.1.1 Vue.js

The primary goal of Vue is to easily create interfaces which react on data changes, so when a Vue instance is created you need to define a template and a model. Inside these templates you can interpolate properties with curly braces like you do in popular templating engines or use attributes.

```
<div id="#view">
  <!-- string interpolation -->
  <h1>{{ name }}</h1>
  <!-- binding syntax -->
  <p v-text="description"></p>
</div>
```

See https://vuejs.org/v2/guide/ for an introduction to Vue.js 2.x.

4.1.2 Single File Components

A Widget is developed as a “Single File Component” with a .vue extension. Each *.vue file consists of three types of top-level language blocks: <template>, <script>, and <style>, and this allows you to put everything regarding a component in one place.

```
<template>
  <div>
     <!-- Write your HTML with Vue in here -->
  </div>
</template>

<script>
  module.exports = {
     // Write your Vue component logic here
  }
</script>

<style scoped>
  /* Write your styles for the component in here */
</style>
```

The <style> tag must include “scoped” as this ensures that the CSS will only apply to the current component.

Below is an example of a very simple Widget – it has a data model that only contains a msg, a template that renders a div tag and puts the msg from the model inside it and finally a CSS class that applies a red color.

```
<template>
  <div class="example">{{ msg }}</div>
</template>

<script>
module.exports = {
  data () {
    return {
      msg: 'Hello world!'
    }
  }
}
</script>

<style scoped>
.example {
  color: red;
}
</style>
```

The Aula platform takes care of processing the files, so Widget Suppliers only need to upload a .vue file to test it in action.

There are several Vue.js text editor plugins available that we encourage Widget Suppliers to use (Emacs, Sublime Text, Visual Studio and Visual Studio Code, Intellij and others).

See https://vuejs.org/v2/guide/single-file-components.html for more information about Single File Components.

4.1.3 Iframe widgets

Iframes offer the distinct advantage of being framework agnostic. Developing an Iframe widget provides the widget supplier with the flexibility to choose a tech stack that aligns with their specific requirements, while seamlessly integrating with Aula. This allows for a customized and tailored approach to meet the widget supplier's needs while maintaining compatibility with the Aula platform.

4.2 Input parameters

Every Widget component instance has its own isolated scope. This means you cannot (and should not) directly reference parent data in Aula from a Widget template.

Context data will be passed down to a Widget component using props, but Widgets cannot send updated values back to the Dashboard.

A Widget component needs to explicitly declare the props it expects to receive using the props option:

```
<template>
  <div>
     <!-- Write your HTML with Vue in here -->
  </div>
</template>

<script>
module.exports = {
  props: {
    // basic type check (`null` means accept any type)
    propA: Number,
    // multiple possible types
    propB: [String, Number],
    // a required string
    propC: {
      type: String,
      required: true
    },
    // a number with default value
    propD: {
      type: Number,
      default: 100
    },
    // object/array defaults should be returned from a
    // factory function
    propE: {
      type: Object,
      default: function () {
        return { message: 'hello' }
      }
    },
    // custom validator function
    propF: {
      validator: function (value) {
        return value > 10
      }
    }
  },
  data () {
    return {
      msg: 'Hello world!'
    }
  }
}
</script>

<style scoped>
  /* Write your styles for the component in here */
</style>
```

Aula will support a number of properties, that can be freely used by Widget Suppliers.

The currently planned properties for the four types of Widgets are:

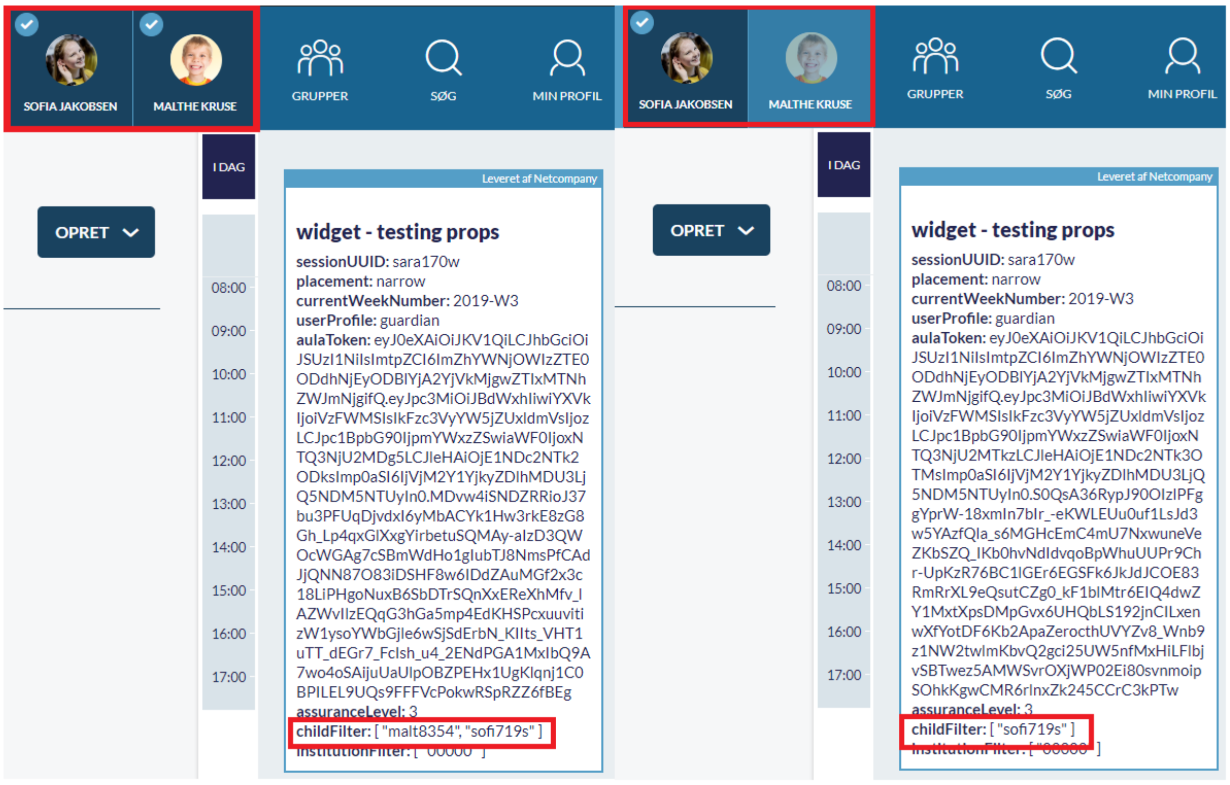
Property name	Type	Description	Iframe Widgets	Normal Widgets	Secure Widgets	SSO Shortcuts
axios	Function	Axios instance, see section 4.4.2.	X	X	X	
sessionUUID	String	Used as a correlation ID in case there is a need to investigate system or user behavior across Aula and external systems. Must be sent as a parameter in all Widget Supplier web service so that	X	X	X	X

		the Widget Suppliers can log user actions in such a way that behavior can be traced across systems.				
placement	String	Which area the Widget has been inserted into – can be either full, medium or narrow.	X	X	X	
currentWeekNumber	String	If the Dashboard allows you to page forward/back between week numbers, this property will contain the currently displayed week number. Can be used to see calendars side-by-side across Widgets. Format: 2018-W48	X	X	X	X
isMobileApp	Boolean	True if Widget is shown in Mobile App and false otherwise (i.e. false on a Mobile browser).	X	X	X	X
getAulaToken (portal)	Function	Helper function that generates a short-lived Aula token, that the Widget Supplier can use to authenticate towards their own Web API (ensuring that the Web API can only be called by Aula Widgets). The generated token is a Base64 encoded, signed JSON Web Token (JWT) that contains • the UNI-Login User ID of the user logged into Aula (sub claim) • the assurance level of the login (assurancelevel claim) • the user session UUID (session_uuid claim)	X		X	X Sent as AulaToken parameter.

		• the token expiration time (exp claim) • the token issuer (iss claim with the value "Aula") • and the system ID for the Widget Supplier (aud claim).				
getAulaToken (noticeboard)	Function	Helper function that generates a short-lived Aula token, that the Widget Supplier can use to authenticate towards their own Web API (ensuring that the Web API can only be called by Aula Widgets). The generated token is a Base64 encoded, signed JSON Web Token (JWT) that contains • The institution code for the notice board (sub claim) • The id of the noticeboard (noticeboardid claim) • the session UUID (session_uuid claim) • the token expiration time (exp claim) • the token issuer (iss claim with the value "Aula") and the system ID for the Widget Supplier (aud claim).	X		X	X Sent as AulaToken parameter.
assuranceLevel	Integer	Is 2 for normal username/password login and 3 for two-factor login (NemID or similar)	X		X	X
institutionCode	String	Only for notice board. Institution code of the		X		

		institution where the notice board is set up.				
userProfile	String	The currently active user profile (child, employee, guardian, otp). This is especially relevant for users that can be both Teacher and Guardian in Aula.	X	X	X	X
childFilter	String array	If the Widget should only display content for specific children, this property will contain the UNI-Login User Id of the chosen children. This can happen if the Aula user (a parent) is filtering his view.	X		X	X
institutionFilter	String array	If the Widget should only display content for specific institutions, this property will contain the UNI-Login institution number of the chosen institution. This can happen if the Aula user is filtering his view or if the Widget has been inserted by a specific institution, rather than as a central Widget or across a whole Municipality.	X		X	X
group	String	If the Dashboard is only showing content for a specific UNI-Login group (a class, year, SFO, Team etc.), this property will contain the UNI-Login Group Id.	X		X	X
triggerStepUp	Function	Helper function that triggers Aulas stepup notification, prompting the user to stepup their assurance level. Can be used by Secure and iframe widgets if the user is not currently logged in with high enough assurance level.	X		X	

		The user can accept the notification, which redirects them to their stepup provider (e.g. UniLogin). The user can also decline the notification, with no further action taken. Users below age 15 cannot step up and will not have the option to accept the notification. The user might not want to step up at all, so widgets should not use this function in start-up/mount, but rather trigger the function upon user interaction with the widget.				
notifications	Array	List of notifications created by the Widget Supplier.	X		X	
deleteNotifications	Function	Helper function that triggers Aulas delete notification(s) service. Can only be used by secure and iframe widgets.	X		X	
setIframeHeight	Function	Helper function that updates the height of the iframe on Aula.	X			
getProps	Function	Helper function to manually retrieve updated Aula Props. *Note: Aula token is exempt from props	X			



widget - testing props

sessionUUID: sara170w
placement: narrow
currentWeekNumber: 2019-W3
userProfile: guardian
aulaToken: eyJ0eXAiOiJKV1QiLCJhbGciOiJSUzI1NiIsImtpZCI6ImZhYWNjOWIzZTE0ODdhNjEyODBlYjA2YjVkMjgwZTIxMTNhZWJmNjgifQ.eyJpc3MiOiJBdWxhIiwiYXVkIjoiVzFWMSIsIkFzc3VyYW5jZUxldmVsIjozLCJpc1BpbG90IjpmYWxzZSwiaWF0IjoxNTQ3NjU2MDg5LCJleHAiOjE1NDc2NTk2ODksImp0aSI6IjVjM2Y1YjkyZDlhMDU3LjQ5NDM5NTUyIn0.MDvw4iSNDZRRioJ37bu3PFUqDjvdxI6yMbACYk1Hw3rkE8zG8Gh_Lp4qxGlXxgYirbetuSQMAy-aIzD3QWOcWGAg7cSBmWdHo1glubTJ8NmsPfCAdJjQNN87O83iDSHF8w6IDdZAuMGf2x3c18LiPHgoNuxB6SbDTrSQnXxEReXhMfv_IAZWvIIzEQqG3hGa5mp4EdKHSPcxuuvitizW1ysoYWbGjle6wSjSdErbN_KIIts_VHT1uTT_dEGr7_FcIsh_u4_2ENdPGA1MxIbQ9A7wo4oSAijuUaUlpOBZPEHx1UgKlqnj1C0BPILEL9UQs9FFFVcPokwRSpRZZ6fBEg
assuranceLevel: 3
childFilter: ["malt8354", "sofi719s"]
institutionFilter: ["00000"]

widget - testing props

sessionUUID: sara170w
placement: narrow
currentWeekNumber: 2019-W3
userProfile: guardian
aulaToken: eyJ0eXAiOiJKV1QiLCJhbGciOiJSUzI1NiIsImtpZCI6ImZhYWNjOWIzZTE0ODdhNjEyODBlYjA2YjVkMjgwZTIxMTNhZWJmNjgifQ.eyJpc3MiOiJBdWxhIiwiYXVkIjoiVzFWMSIsIkFzc3VyYW5jZUxldmVsIjozLCJpc1BpbG90IjpmYWxzZSwiaWF0IjoxNTQ3NjU2MTkzLCJleHAiOjE1NDc2NTk3OTMsImp0aSI6IjVjM2Y1YjkyZDlhMDU3LjQ5NDM5NTUyIn0.S0QsA36RypJ90OIzIPFggYprW-18xmIn7bIr_-eKWLEUu0uf1LsJd3w5YAzfQIa_s6MGHcEmC4mU7NxwuneVeZKbSZQ_IKb0hvNdIdvqoBpWhuUUPr9Chr-UpKzR76BC1IGEr6EGSFk6JkJdJCOE83RmRrXL9eQsutCZg0_kF1blMtr6EIQ4dwZY1MxtXpsDMpGvx6UHQbLS192jnCILxenwXfYotDF6Kb2ApaZerocthUVYZv8_Wnb9z1NW2twImKbvQ2gci25UW5nfMxHiLFIbjvSBTwez5AMWSvrOXjWP02Ei80svnmoipSOhkKgwCMR6rInxZk245CCrC3kPTw
assuranceLevel: 3
childFilter: ["sofi719s"]
institutionFilter: ["00000"]

Figure 15: childFilter and institutionFilter properties are based on UI Filters on top menu

Note, that for SSO Shortcuts the parameters will all be included as POST, or as GET for app, parameters to the SSO Shortcut URL.

The list of supported properties may grow during design and development of Aula and also afterwards as new Widget requirements and ideas turn up.

Note, that if the Dashboard changes a property value, the value will change inside the Widget as well. To react on property changes, you can use Vue.js features such as Computed and Watched Properties (see https://vuejs.org/v2/guide/computed.html).

For iframe widgets, functions are exposed as post message functions and returned as props.

Example:

```
// Send Request
parent.postMessage(
  {
    request: ‘deleteNotifications’,
    metadata: {
       notificationIds: [1,2,3],
    }
  },
  ‘https://aula.dk’,
);
```

```
// Retrieve information
window.addEventListener('message', (event) => {
    if (event.origin !== ‘https://aula.dk’) { // Alternatively, use parent.window !== event.source
for a more generic approach
        return;
    }
    state.myData = event.data;
});
```

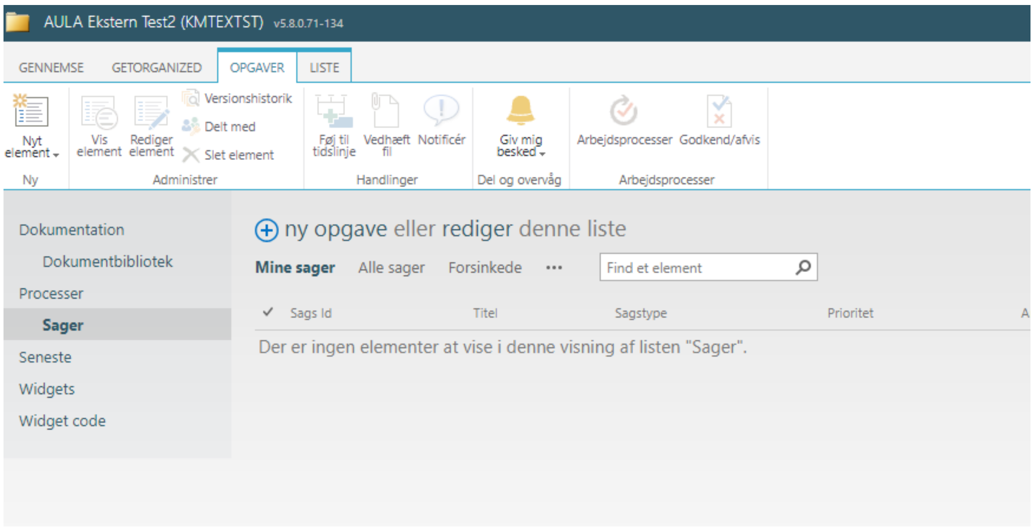
In addition to the input properties, a Widget Supplier can use window.navigator from javascript to detect userAgent etc.

4.3 How to use toolkit

This page will work 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 code and metadata.

4.3.1 Applying for new or change in widget

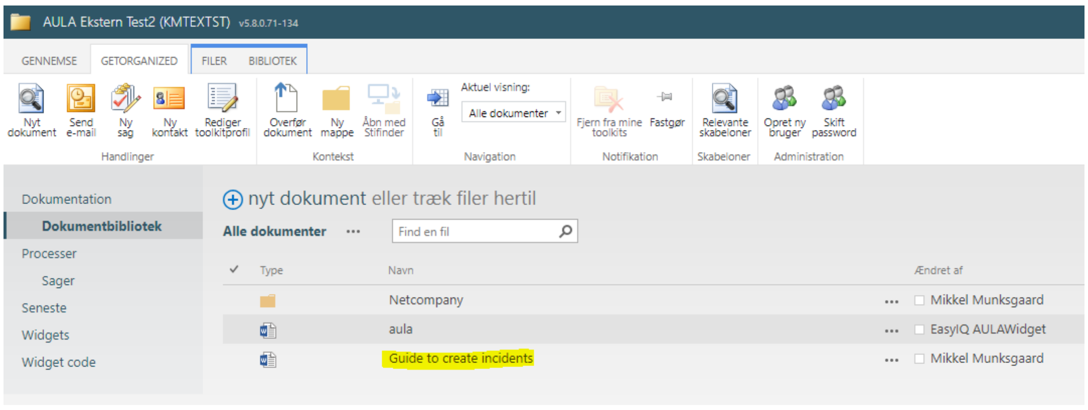
All the applications described in 3.2.2, 3.3.2 and 3.5.2 are done within the Cases(Sager) menu. At the top of this menu page the application can be started by pressing the "new assignment(ny opgave)". This open the formula used for both Service Request, which will be the focus of this section, and Incident reporting.



Der er ingen elementer at vise i denne visning af listen "Sager".

Figure 16: Toolkit cases (Sager)

For a guide for creating incident report consult the "Guide to create incidents" found within the document library on the page.



Type	Navn	Ændret af
	Netcompany	Mikkel Munksgaard
	aula	EasyIQ AULAWidget
	Guide to create incidents	Mikkel Munksgaard

Figure 17: Guide to create incidents

When creating a new case the default CaseType is a Service Request. These can be used to create a service request from either the Service Desk, Operations or Consulting Team. Just below the CaseType is a dropdown menu called Service Catalog (Servicekatalog) of the widget standard services described in chapter 3. When selecting a standard services the formula will set Title and Team and fill in the Description (Beskrivelse) with the needed fields for the specific request. When a request have been sent it will become and entry on the list and the status of it can be followed.

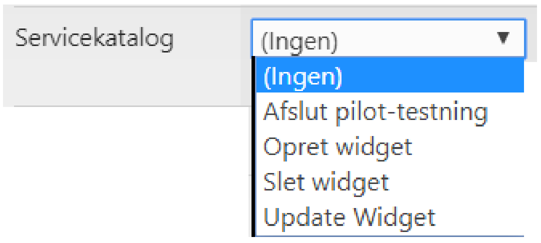


Figure 18 Service Catalog from Aula External Test 2

The Status for a request are:

- 10 – New
- 31 – On Going
- 60 – Solved
- 80 – Awaiting customer
- 90 – Closed
- 91 – Rejected
- 93 - Cancelled

When a widget is approved the metadata and widget code is put into its respected lists on the system. Here the data what is the same for both is placed on the list on the Widgets page. While the code and metadata that can be unique is placed on the list on the Widget code page. The columns for these lists are descripted in detail underneath in 4.3.2. Under the Widgets menu which widget that is used as release and pilot is specified and the approval status of the widget can be seen.

The statuses are as followed

- 1 – New
- 30 – Ready for review
- 35 – Approved
- 40 – Ready for rollout
- 70 – Rejected with few errors
- 90 – Rolled out
- 99 – Rejected with significant errors

Each of these will the changed according to the procedures descripted in 3.2 and 3.4.

4.3.1.1 Notice board widgets

Notice boards are usually placed in open locations and sensitive information should not be disclosed in a widget placed on a notice board. Widgets for notice boards can be of the types Normal or Noticeboard. Widgets of type “Noticeboard” will be able to issue an aula token to allow you to verify the request is originating from a notice board configured in Aula. Remember to validate both the token and the institution code claim in the token.

When registering/altering your widget in the Toolkit, you must mark it as being applicable for notice boards (see section 4.3.2). This is important as it determines whether the institution administrators will be able to select the widget when setting up their notice boards.

4.3.2 Widget data specifications

Overview and description of the columns present on Widgets and Widget code. All except widget status and the cross references are transferred over to Aula.

Name	Type	Description
Widgets		
WidgetId	Automated	This values is assigned based on the Widgets case number and is used in Aula to identify the widget
Name	Single line string	Name of the widget
Description	Multi line string	A Description of the widgets function
Url	Single line string or empty	For SSO types this will be the supplied link for the others this will contain the folder location of the widget code
Icon	Single line string or empty	A single sting containing the file location of the image. Can reference an external URL or a location in aula. Image needs to be in a PNG format and around 200 by 200 pixels. If it is a local file, it should be uploaded along with the widget code. Format examples: “/static/widgets/W0001V0001.png” or “https://url.dk/Widget_Icon.png”
WidgetSupplier	Automated	Name of supplier
Kontaktperson(Contactperson)	Automated	Contact person from the supplier
WidgetStatus	Handled by Aula’s Central Administration	Shows the current status of the widget from the following options: - 01 – New - 20 – Ready for review from Central Administration - 30 – Ready for review from Widget Board - 40 – Ready to rollout - 70 – Rejected with few errors

		- 90 – Rolled out - 99 – Rejected with significant errors
WidgetType	Selection	Setting which of the types the widget is
ReleaseVersion	Selected from menu	Select which widget code from the “Widget code” that is sent to Aula as the Release version
ReleaseVersion:Id	Automated	Id based on entry number within “Widget code”
PilotVersion	Selected from menu	Select which widget code from the “Widget code” that is sent to Aula as the Pilot version
PilotVersion:Id	Automated	Id based on entry number within “Widget code”
Widget Code		
Name		
WidgetVue	Multi line string	The code of the widget
WidgetNotificationVue	Multi line string	The code used for the widget notification
CanBePlacedOnGroups	Boolean	Determines if the widget can be shown on groups
CanBePlacedInsideModule	Boolean	Determines if the widget can be shown inside modules
CanBePlacedOnFullPage	Boolean	Determines if the widget can be shown as a full page widget
CanBePlacedOnNoticeBoard	Boolean	Determines if the widget can be shown on notice boards
SupportsTestMode	Boolean	Determines if the widget can support test mode
WidgetVersion	Single line string	Gives version of widget Format example: v1.3
Widget	Selected from menu	Select which Widget in the “Widgets” menu the code relates to
Widget:WidgetId	Automated	WidgetId for selected widget

4.4 How to use your own Web API

In order to show dynamic data, Widget Suppliers need to call their own REST based Web API exposing the relevant web services that return JSON data (not HTML).

In all web service calls, Widget Suppliers must send the session UUID parameter and use this to log user actions in their own systems (in order to be able to trace behavior across systems).

4.4.1 Web API Security

All Web services must use HTTPS, both to ensure encryption of tokens and data and to ensure that Aula and Aula users always obtain content from the correct source. Do consider the following when developing the web API for a widget:

- Aula will attempt to upgrade any insecure requests to HTTPS, and any attempts to connect to insecure API's will therefore fail.
- Ensure that the web API serve valid certificates, preferably satisfying Certificate Transparency requirements.
- Aula will attempt to validate the revocation status of certificates served by external web APIs and will reject HTTPS connections based on revoked certificates. Make sure to monitor the revocation status of the certificates served by your widget web API.

Widgets that use information about the user (UNI-Login ID etc.) must additionally:

- Obtain an Aula Token (JSON Web Token) using the getAulaToken function and send token as authentication mechanism (typically as Authorization header using the Bearer schema).
- Perform security validation in their Web API (validate that the token is signed by Aula, is issued for their System ID and has not yet expired) and extract UNI-Login ID of the user or institution code for noticeboard, in order to deliver personalized data.
- Specific for Secure widgets, regardless of other context parameters sent to the Widget Suppliers web service, the service must never return data that should not be accessible to the UNI-Login ID and assurance level provided in the Aula Token.

For further information about JSON Web Tokens, see https://jwt.io/.
Depending on what web API you have there can be different ways to verify the JWT token sent from Aula as the Aula token. This verification should be towards the certificate found on the widget toolkit on the "Produktions certifikat" page.

For more info on JWT tokens and what libraries to use for your web API's language check https://jwt.io/libraries

4.4.2 REST API helper library

Aula provides the axios library to make it easy for Widget Suppliers to consume their own REST Web API.

In order to use Axios, you only have to write "this.axios" in the <script> section of the .vue file to access the local instance of the Axois library.

Axios uses promises by default, and makes it easy to use asynchronous web service calls and do error handling as illustrated by the example below.

```
<template>
  <ul v-if="posts && posts.length">
    <li v-for="post of posts">
      <p><strong>{{post.title}}</strong></p>
      <p>{{post.body}}</p>
    </li>
  </ul>

  <ul v-if="errors && errors.length">
    <li v-for="error of errors">
      {{error.message}}
    </li>
  </ul>
</template>

<script>
module.exports = {
  data: () => ({
    posts: [],
    errors: []
  }),

  // Fetches posts when the component is created.
  created() {
    this.axios.get(`http://jsonplaceholder.typicode.com/posts`)
    .then(response => {
      // JSON responses are automatically parsed.
      this.posts = response.data
    })
    .catch(e => {
      this.errors.push(e)
    })

  }
}
</script>
```

Axois supports configuring the instance which can be used to e.g. set a base URL and the AulaToken header on all requests:

```
// Alter defaults on the instance
this.axios.defaults.baseURL = 'https://api.example.com';
this.axios.defaults.headers.common['Authorization'] = AUTH_TOKEN;
```

Axios can do both GET, POST, PUT, DELETE etc. and also supports performing concurrent requests as well as adding interceptors to requests/responses.

As Widget Supplier, you are not allowed to modify the global Axios default configuration or add interceptors to the global axois instance. Widgets that make use of the global Axios instance in this way will not be approved.

See https://github.com/mzabriskie/axios for more information about how to use Axios.

4.5 Widget notifications

This section is a technical description of how to send, handle and delete notifications for secure widgets and iframe widgets. For a description on what notifications in Aula entails, see section 2.3.1.3.

4.5.1 Sending notifications

Widget Suppliers can send notifications through the external widget API. The API is not directly accessible from the widget, but instead must be called from the Widget suppliers' own system. The purpose of notification is to notify a user about some change in the systems of the widget vendor, which may not be when the user is looking at the widget.

A widget supplier must apply for access to the API before they can send notifications to users. The widget supplier will have two main functions for sending notifications:

- specifically sending a single notification to one user, or
- sending a notification to all members of a group.

The notification is handled with a “notification id” which the widget supplier must provide in the request, and use to identify the notification, as it is the responsibility of the widget supplier to delete the notifications correctly. Examples of when a notification should be deleted includes:

- after a specific time,
- when the user navigates to specific locations in the widget or
- when the user interacts with or views the secure widget.

4.5.1.1 Applying for API key

As a Widget Supplier you can request an API key and IP-Whitelisting through a case in the toolkit, see section 4.3 for a guide on how to create a case in the toolkit. The API key is used to uniquely identify the Widget, thus the Widget Supplier must request one key per Widget they want to send notifications from. Make a case as you would if making a change request to a widget but ask for access to send notifications from your secure widget. Please include a short description of why you need notifications for your secure widget.

All API keys are limited to a fixed number of notifications they can send, the default amount is 1000 notifications per widget per hour.

Widget suppliers can also apply for a higher limit if that is needed. Provide the increased limit in in your initial request for an API key or make a new case in the toolkit as you would when requesting an API key. In the case you need a higher limit, it is important that you describe why this is needed, and how your secure widget uses and deletes the notifications that you send out.

4.5.1.2 Notification id

A notification is not sent from a secure widget, but instead triggered by calling Aula´s external widget API. When using the API the Widget Suppliers must specify a “notification id” that the Widget Supplier can use to identify the notification later when handling and deleting the notifications.

The “notification id” of all active notifications will be available for the secure widget, through the widget interface. This means that if the secure widget has 1 “notification id" available, the user will see a badge in aula with the number 1 inside, thus it is important that the widget suppliers handle the “notification id”, if not implementing a simple deletion rule for notifications, so that the badge won´t be shown indefinitely. If it is not deemed necessary to delete notification badges after specific user actions, based on the individual notification, a simple rule could be used, example of a simple rule: Whenever the user views the widget or click on a specific tab, the secure widget deletes all notification badges that are active for the current user.

4.5.1.3 Sending notifications to single users

Through the external widget API, widget suppliers can send either “badge” or “push” notifications to specific users, see section 2.3.1.3 for a detailed description of what the user will see in Aula with the 2 different notification types. It is recommended to use both either “badge” or both, as a push notification without a badge might be confusing for the user.

A widget supplier must specify the specific user they want to send a notification to, by sending “Institution Code”, “Portal role” and “User Id”. All those values are available through the widget interface as a secure widget.

It is not possible to send notifications directly through the secure widget, this must be handled by the widget suppliers separately from the secure widget itself.

4.5.1.3.1 Request format

The API is accessed through this URL “widget-api.aula.dk/userNotification", the communication is handled through HTTPS with a JSON formatted data model, as a standard HTTP PUT request. The API key should be sent as a “x-api-key-header”.

The following table describes the data model the API expects when sending notifications to a single user.

Name	Type	Cardinality	Description
userId	String	1	The Unilogin username, of the user that you want to send the notification to.
institutionCode	String	1	The institution code of the user.
notificationId	String	1	“Id” that will be returned to the Widget when the user navigates to the widget.
portalRole	String	1	The type of user you want to send the notification to. Possible inputs: ["Employee", "Guardian", "Child"]. Note that when targeting Otp profiles, you should use the “Guardian” as the portalRole.
notificationTypes	String[]	1…2	The kind of notification the user receives. Possible inputs: ["Push", "Badge"]

4.5.1.3.2 Example

An example of the PUT request for sending a badge and push notification to a single child user identified by “esbe3854” on the institution “00000”.

```
PUT widget-api.aula.dk/userNotification HTTP/1.1
Content-Type: application/json
x-api-key: YkhKnW2WhMyHnQUxvi7emBD8nkdWhYfBaks7FNkFhctL3
Accept: */*
Accept-Encoding: gzip, deflate, br
Connection: keep-alive
Content-Length: 169

Body:
{
        "UserId": "esbe3854",
        "institutionCode": "00000",
        "notificationId":"SmoketestUser-1",
        "portalRole": "child",
        "notificationTypes": ["Push","Badge"]
}
```

Any HTTP response that is not “200 OK” will result in no notifications being sent.

4.5.1.4 Sending notifications to Uni groups

Through the external widget API, widget suppliers can send either “badge” or “push” notifications to all parents, children, or employees of a Uni group, see the section 4.5.1.3 above for recommendations on which notification type to send. The notification will function the same as a notification send to a single user, and the user can’t tell the notifications apart.

A widget supplier must specify the specific Uni group, they want to send a notification to, by sending “Institution Code” and “Uni group Id”. Additionally, a list of “portal roles” is used to specify if notifications should be sent to all Children, Employees and/or parents in the group, it is possible to choose one, two or all three role-specific sub-groups of the specified group.

If an API call fails when sending notifications to groups, none of the notifications will be sent. It will always send all or none of the notifications.

4.5.1.4.1 Request format

The API is accessed through this URL “widget-api.aula.dk/groupNotification", the communication is handled through HTTPS with a JSON formatted data model, as a standard HTTP PUT request. The API key should be sent as a “x-api-key-header”.

The following table describes the data model the API expects when sending notifications to all members in a group, sub-grouped by “Employees”, “Guardians” or “Children”.

Name	Type	Cardinality	Description
uniGroupId	String	1	The Unilogin Group Id, of the group that you want to send the notifaction to.
institutionCode	String	1	The institution code of the group.
notificationId	String	1	“Id” that will be returned to the Widget when the user navigates to the widget.
portalRole	String[]	1…3	Which type of users you want to send the notification to. Possible inputs: ["Employee", "Guardian", "Child"]. Note that setting portalRole to “Guardian” both Guardians and Otp´s will receive notifications.
notificationTypes	String[]	1…2	Possible inputs: ["Push", "Badge"], could be one or both.

4.5.1.4.2 Example

An example of the PUT request for sending a badge and push notification to all users of a group identified by “uid2” on the institution identified by “00045”.

```
PUT widget-api.aula.dk/groupNotification HTTP/1.1
Content-Type: application/json
x-api-key: YkhKnW2WhMyHnQUxvi7emBD8nkdWhYfBans7FNkFhctL3
Accept: */*
Accept-Encoding: gzip, deflate, br
Connection: keep-alive
Content-Length: 211

{
        "uniGroupId": "uid2",
        "InstitutionCode": "00045",
        "NotificationId": "Group-notification-3",
        "portalRoles": ["employee","guardian","child"],
        "NotificationTypes": ["Badge", "Push"]
}
```

Any HTTP response, to this request, that is not “200 OK” will result in no notifications being sent. This means if you are about to send notifications to a group with 50 members and you get “Rate limit reached”, it could potentially be possible to send a single notification to 49 people right after.

4.5.1.5 HTTP responses and error codes

After sending the HTTP request meant to send a notification to either a single user or a group, the REST API will return a HTTP response. The response will be marked with the HTTP code “200 OK” if the notification is sent, thus any other response will result in no notifications being sent. The following responses can be expected:

A “200 OK” HTTP response:

```
HTTP/1.1 200 OK
Content-Type: application/json
Content-Length: 43
Connection: keep-alive

Body:
{
        "Message":"Success",
        "NotificationsSent":2
}
```

A “429 Too Many Requests” HTTP response, indicates that the widget has reached its limit, and no more notifications can be sent. The limit is set per hour, so the notifications can be sent later.

```
HTTP/1.1 429 Too Many Requests
Content-Type: application/json
Connection: keep-alive

Body:
{
        "message":"The limit on amount of notifications has been reached",
}
```

A “400 Bad Request” HTTP response, can be malformed request or user cannot be found. If the user cannot be found, no specific information is given as to why. An example of a malformed request, missing the “portalRoles” field:

```
HTTP/1.1 400 Bad Request
Content-Type: application/json
Connection: keep-alive

Body:
{
        "message":"Missing required fields",
        "details":”portalRoles",
}
```

4.5.2 Handling notifications

This section aims at describing the dataset and usage of widget notifications.

The creation and deletion of notifications are the sole responsibility of the Widget Supplier. Aula does not control or have any influence over the content of notifications, though notifications older than 3 months will be deleted.

Note that there might be a difference between the badge notifications that is present in Aula, and the notifications tracked by the Widget Supplier. For example, a notification could be expired in Aula, or if the creation of the notification or multiple notifications fails, then no notifications will be created in Aula. In both those cases, the Widget Supplier might have saved and be actively tracking notifications that does not exist for the user in Aula. Therefore, the Widget Supplier is recommended to cross reference badge notifications tracked by the Widget Supplier with the active badge notifications that can be found in the Secure/Iframe Widgets data supplied by Aula. The next section will describe how to find the active list of badge notifications supplied by Aula.

4.5.2.1 Using Badge notifications

Badge notifications are lists of objects containing the following properties.

```
{
  widgetNotificationId //Represents the unique ID generated by the widget vendor
  notificationMessage //Represents the message generated by Aula, currently this information is not
relevant for the widget supplier.
  triggered //Represents a date string of when the notification was created Ex.2023-01
16T04:09:41+00:00
  institutionCode //Represents the institution code the widget vendor created the notification for
}
```

Usage in the DOM can be achieved like this.

```
<template>
<div>
 <div v-for=”notification in notifications”>
  Besked {{notification.widgetNotificationId}}: {{ notification.notificationMessage }}
 </div>
</div>
</template>
```

4.5.2.2 Deleting badge notifications

Aula will show a badge with a number corresponding to the sum of active badge notifications, if the widget supplier deletes one badge notification, the number will be reduced by one. If there are no badge notifications, Aula will hide the badge completely. As a Widget Supplier you must delete all badge notifications correctly, if the user should not be notified about new information in the Widget. Thus, it is important that the Widget Supplier ensures that the active badge notifications supplied by Aula is correct according to the list of badge notification tracked by the Widget Supplier, to ensure a good user experience. Alternatively, a Widget Supplier might delete all notifications when the Widget is loaded, as a safe way to ensure that notifications will not result in a bad user experience.

When a badge notification is deleted, the notification will still be showed in Aula, until the user refreshes the page or navigates to a new page in Aula.

When a Widget Supplier wants to delete a badge notification or multiple badge notifications, Widget Suppliers can use the deleteNotifications function.

Example1: Delete all notifications.

```
const notificationIds = this.notifications.map(notification =>
notification.widgetNotificationId);
this.deleteNotifications(notificationIds);
```

It is important to note that deleteNotifications only accepts a list of widgetNotificationIds, Following example illustrates how to delete a single badge notification.

Example2: Delete a single notification.

```
<template>
<div>
 <div v-for=”notification in notifications”>
  Besked {{notification.widgetNotificationId}}: <button
@click=”removeNotification(notification)”>delete</button>
 </div>
</div>
</template>
<script>
export default {
 props: {
  notifications: Array,
  deleteNotifications: Function,
 },
 methods: {
  removeNotification(notification) {
   this.deleteNotifications([notification.widgetNotificationId]);
  }
 }
}
</script>
```

4.6 Using the Aula Token for Single-Sign-On

The Aula Token can be used to implement Single-Sign-On, so that a Widget can open up new browser windows with full access to an external system.

If this is done via an SSO Shortcut Widget, Aula takes care of generating the necessary HTML markup to do the POST of the Aula Token. Do to limitations in iOS webviews, SSO shortcuts will be send as GET from the app.

If Widget Suppliers want to embed similar SSO links inside their own Widgets, the recommended approach is to:

- Use a HTML Form (with a target=”_blank” or a named target to ensure it opens in a new tab/window) and generate and include the short-lived Aula Token in a hidden input field when the user sends the form.
- If necessary, other Widget parameters can be included in the paramteres (e.g. userProfile)

Regardless of whether the SSO link is from a SSO Shortcut or a custom Widget, the URL on the Widget Suppliers website that receives the sign-on token must

- Perform the same JSON Web Token validation as described in Section 4.4.1 (validate that the token is signed by Aula, is issued for their System ID and has not yet expired).
- Log in the user based on the UNI-Login Id in the token.
- Log the received session UUID from the token together with the local session Id in order to be able to trace behavior across systems.

4.7 Access to 3rd party resources

For security reasons, Widget Suppliers are not allowed to access external JavaScript libraries, CSS files or similar content that could affect the look or behavior of the Aula solution. Any attempts to load external resources not explicitly allowed by Aula, will be blocked by Aulas Content Security Policy.

The only external resources you are allowed to use in a .vue Single File Component are:

- The Widget Supplier’s own REST Web API that returns JSON data
- Images used in the HTML markup inside template part of the .vue Single File Component

Aula will supply a number of predefined external resources that Widget Suppliers can reference inside Widgets:

- BootstrapVue, Bootstrap 4 integrated with vue.js (https://bootstrap-vue.js.org/ - version 2.0.0-rc.15)
- Element UI (https://element.eleme.io/#/en-US)
- Font Awesome, scalable vector icons that can be customized via CSS (http://fontawesome.io/)

- Axios, the REST API library mentioned above (https://github.com/mzabriskie/axios)
- Moment,js, a JavaScript library for handling dates and times (https://momentjs.com/)

More predefined external resources will be added over time.

If Widget Suppliers want to use other external resources, they can test the 3rd part resources in the Test environment. Subsequently, they can then apply to have the new external resources added to the Aula production environment, but this requires approval by KOMBIT as part of the Widget Approval described in section 0 and 3.34.

4.7.1 Iframe widgets

Iframe widgets have the flexibility to utilize 3rd party dependencies within the widget itself if they do not involve communication with other domains. Any attempts to communicate with external domains not whitelisted by Aula will be blocked by Aula's Content Security Policy.

4.8 Limitations

For both security and stability reasons, Widgets are not allowed to access Aula data (other than the input parameters described in section 4.2) or influence the layout of other parts of Aula.

This introduces the following limitations when developing Vue.js Single Page Component Widgets:

- All HTML content must be part of the <template> section of the .vue file.
  - You are not allowed to inject dynamically created HTML, e.g. by loading it from a web service and injecting it into your Widget using innerHtml, v-html or similar approaches.
  - Dynamic content should instead be handled via standard Vue.js model binding in the HTML template.
- All anchor tags (links) must use a target="_blank" or use a named target in order to open in a new tab/window.
- The JavaScript inside the Widget is not allowed to use JQuery or to access or manipulate the DOM, window.location, window.history or any other access to window except window.navigator (for accessing information about the browser).
  - This also excludes access to this.$parent and this.$refs in Vue.js.
- All access to Web API's must use asynchronous calls (e.g. via Axios promises) to keep Aula responsive.
- The <style> tag must include the "scoped" attribute to limit CSS styling to the component itself.
- Do not access or manipulate the Aula-specific cookies or request headers. This includes but are not limited to the host header and cross-site request forgery tokens.

The majority of these technical limitations will be validated automatically whenever Widget Suppliers update the source code of a Widget in the Aula Widget Test Environment. Widget Suppliers will therefore be prevented from making simple technical mistakes that lead to not getting their Widget approved.

4.8.1 Mobile App Limitations

If the user clicks a link or similar, which navigates away from the widget's frontpage, it cannot be expected that the device's back button can be used to navigate back. If a 'back' functionality is required, then this should be built into the widget. Otherwise the user can always navigate away and back to the widget where it will then be reloaded from the widget's frontpage.

4.9 Best practices

The following best practices are recommended in order to develop Widgets that perform well and have good usability.

4.9.1 Performance

Aula Widgets are executed in the same way as the modules inside Aula itself and in order to achieve a good user experience, it is important that Widgets and their underlying Web API's are optimized for performance as well as perceived performance.

4.9.1.1 JavaScript

The JavaScript code in Widgets is only for "frontend logic", i.e. for coordinating events, executing Web API calls and assigning results of Web API calls to the widget. The JavaScript should not contain business logic or calculations – this should be handled in the Widget Suppliers Web API.

In general, any JavaScript execution (individual event handling) should take less than 100 ms so that Widgets do not end up blocking the users browser (note that Web API calls are performed asynchronously and are not part of the event handling time).

4.9.1.2 Web API

Web API calls used for loading the initial data for the Widget should take less than 1 second.

When further user interaction triggers updates or calculations through Web API, the response time can be longer, but this requires indication of response time as described below.

4.9.2 Iframe widgets

Iframe widgets have the unique benefit of being framework agnostic, which means they can be developed using different frontend frameworks or even without any specific framework. This flexibility allows widget suppliers to choose the tech stack that best suits their needs while integrating seamlessly with the Aula platform.

When preparing Iframe Widgets for production use, it is crucial to optimize them for lazy loading of content. Implementing a chunk splitting algorithm ensures minimal payload size for Iframe widgets.

Additionally, when fetching dynamic components or waiting for APIs, it is essential to provide clear indications to the end-user about ongoing preparation processes. This helps maintain a smooth and informative user experience.

4.9.3 Error handling

Make sure that the JavaScript code handles unexpected errors and shows a meaningful error message to the user:

- All Web API calls should have and Error handler that shows the user that something went wrong and possibly allows them to try again.
- JavaScript code should have try/catch blocks that also show the user that something went wrong.

4.9.4 "Loading" indicators

Any Web API call can end up taking longer than expected (due to latency, network outage, server hickups etc.).

Always make sure that the UI shows some kind of "loading" indicator so that the user can see that something is happening.

The included Font Awesome can be used in the HTML templates to show a spinner while loading data, like in the code below where a spinner is displayed whenever the "loading" field of the model is true:

```
<i class="fa fa-spinner" v-if=”loading” ></i>
```

With Vue.js and Axios, triggering "loading" (and hiding the HTML that shows the result while "loading" field is true) can be done in several ways:

- You can assign the “loading = true;” value before performing the Axios call and then assign “loading = false;” in the callbacks (remember to do it both in the successful callback and the error handler)
- You can implement interceptors that perform the “loading” assignments automatically in all service calls performed through a Axois instance.

4.9.5 Indication of response time

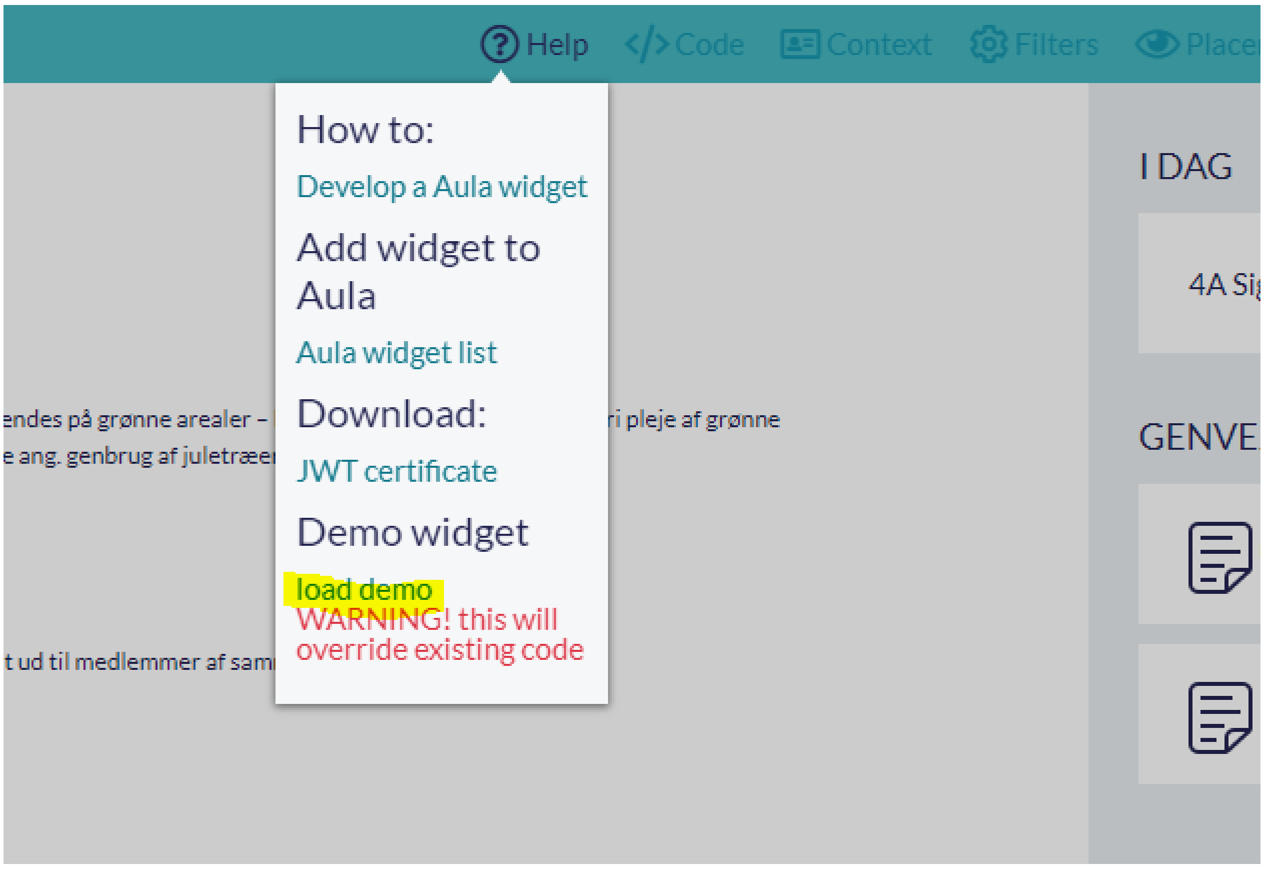
In general, Aula will attempt to give end-users a realistic indicator of expected response time, and Widget Suppliers should endeavor to do the same.

- Actions expected to take less than a second just need to use a “Loading” indicator as described above
- Any action that is expected to take more than one second will signal this to the user with a special indicator for “Longer response time”.
- For any action that is expected to take more than 5 seconds, Aula uses progress indicators to show the user roughly how long the action will take.
    - This can either be done by estimating roughly how long the action normally takes or by implementing the Web API to execute its action asynchronously and immediately return a new URL that can be polled for status (and ultimately will return the result of the action when it completes).

5 Build – good to know

There is an example of how to structure the Vue component that can be loaded in the widget previewer. Look under the help menu for “load demo” and the code section will be filled with an example of a widget.

To develop and preview Iframe widgets, please refer to [WidgetLocalDevelopment].



5.1 Closing tags

In the previewer use closing tags like

```
<div v-if=”someThing” class=”someIcon”></div>
```

Rather than self closing tags like:

```
<div v-if=”someThing” class=”someIcon” />
```

5.2 AulaToken

In the widget previewer getAulaToken returns instantly and therefor code like this is possible:

```
let token = this.getAulaToken();
```

But on Aula the call is not returned instantly, so you should use following paddern:

```
module.exports = {
  props: {
    axios: Function,
    getAulaToken: Function,
    ...
  },
  data: function () {
    return {
      posts: {},
      aulaToken: null,
      ...
    }
  },
  methods: {
    loadStuff() {
      this.axios.get("/images/demo/posts.json")
        .then(response => {
          // JSON responses are automatically parsed.
        })
        .catch(e => {
          console.log(e);
        })
    },
  },
  mounted() {
    this.aulaToken = this.getAulaToken();
  },
  watch: {
    aulaToken: function() {
      if (this.aulaToken != undefined) {
        this.loadStuff();
      }
    },
  },
};
```

5.3 Moment.js

Moment is included as a property, and can be retrieved like this:

```
module.exports = {
  props: {
    ...,
    moment: Function,
    ...
  },
```

And can be used in the code like this:

```
    this.someDate = this.moment().format('MMMM Do YYYY, h:mm:ss a')
```

5.4 HTML component

As we don't allow use of v-html Netcompany has following component available for use to display HTML content:

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

This component allows following HTML tags: <a>, <em>, <p>, <div>, <span>, <strong>, <u>, <ul>, <ol>, <li>, <table>, <thead>, <tbody>, <tr>, <th> and <td> other tags will be stripped. No href or src attributes with javascript is allowed and no use of any onxxx attributes like onclick, onfocus etc. is allowed.

5.5 Using commands from Aula in an Iframe widget

To use functions in Aula like setIframeHight, getAulaToken and getProps from a Iframe widget you will need to make use of the javascript function window.parent.postmessage. As can be seen in the following code example made in Vue.js

The example was made so the Iframe adjusts its size first and then requests an Aula token which is printed in the console as well.

The View of the Iframe is of two buttons that get the props or token and print them in the console.

```
<template>
  <div id="app">
    <p>test</p>
    <button type="button" @click="getProps">Get aula props</button>
    <br />
    <br />
    <button type="button" @click="getToken">Get aula token</button>
  </div>
</template>
```

```
<style scoped>
#app {
  font-family: Avenir, Helvetica, Arial, sans-serif;
  -webkit-font-smoothing: antialiased;
  -moz-osx-font-smoothing: grayscale;
  text-align: center;
  color: #2c3e50;
  margin-top: 60px;
}
</style>
```

```
<script>
export default {
  data() {
    aulaToken: '';
  },
  mounted() {
    //Event Listener for Iframe
    window.addEventListener('message', this.iframeEvent, false);

    //send command to aula to adjust size of iframe window
    window.parent.postMessage(
      {
        request: 'setIframeHeight',
        metadata: {
          height: 575,
        },
      },
      '*'
    );
    this.getToken();
  },
  methods: {
    //only print the event data if the event is sent from the local dev env, testing env or from
aula
    iframeEvent(event) {
      if (
        event.origin !== 'http://localhost:5173' &&
        event.origin !== 'https://localhost:8080' &&
        event.origin !== 'https://aula.dk'
      ) {
        return;
      }
      console.log(event.data); //data is printed it can be used in your widget instead
      if (event.data.type == 'token') {
        // only do something if you are getting a token
        this.aulaToken = event.data.data.token;
      }
    },
    getProps: function (event) {
      console.log('props');
      window.parent.postMessage({ request: 'getProps' }, '*');
    },
    getToken: function () {
      console.log('token');
      window.parent.postMessage({ request: 'getAulaToken' }, '*');
    },
  },
};
</script>
```