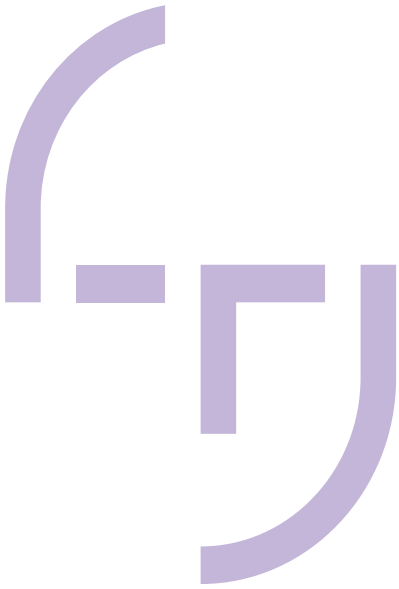
Tampere University of Applied Sciences



Implementing a HbbTV Application Editor

Veera Halonen

BACHELOR'S THESIS
November 2022

Degree Programme in Software Engineering

ABSTRACT

Tampereen ammattikorkeakoulu
Tampere University of Applied Sciences
Bachelor's Degree Programme in Software Engineering

VEERA HALONEN:
Implementing a HbbTV Application Editor

Bachelor's thesis, 53 pages
November 2022

This thesis describes the implementation of a web editor for generating and customizing HbbTV (Hybrid Broadcast Broadband TV) applications for television. The project was commissioned by Sofia Digital and includes a web editor implemented with Vue.js, a Node.js API which serves the configuration generated by the editor, and a Vue-based HbbTV application which receives the configuration and generates its contents based on it. The project applications also utilize certain pre-existing interfaces to supplement their functionality, such as for generating user polls and prize draws for the HbbTV application.

The editor was based on various previously implemented companion applications for Finnish television shows, which also used similar content editors but were always application-specific. This project started as a technology update on those applications but soon evolved into a more generic editor solution allowing even non-developers to easily create and customize simple menu-based HbbTV applications.

The project development is still ongoing and more features are being added, but the core functionality was successfully implemented, and the editor will soon be ready to be deployed in Finland, and later, perhaps internationally. It has already been incorporated in the company’s Sofia Backstage® product family and was also a finalist in the HbbTV Awards 2022 in the category “Best tool or product for HbbTV service development or delivery”.

Key words: hbbtv, hybrid television, javascript, vue

CONTENTS

ABBREVIATIONS

API	Application Programming Interface
CSS	Cascading Style Sheets
DSM-CC	Digital Storage Media Command and Control
DVB	Digital Video Broadcasting
ETSI	European Telecommunications Standards Institute
HbbTV	Hybrid Broadcast Broadband TV
HTML	Hypertext Markup Language
MHP	Multimedia Home Platform
UI	User Interface
VOD	Video on Demand

1 INTRODUCTION

This thesis will examine the development of a web application which can be used to generate and customize simple HbbTV (Hybrid Broadcast Broadband TV) applications. The project includes both the web editor and the HbbTV application it controls, as well as a simple API that serves the data saved through the editor to the HbbTV application.

The application was implemented while working at Sofia Digital Oy, which also commissioned the project. The goal was to produce an application which could be used to quickly produce simple HbbTV applications and allow for easy customization of applications already in production while also being accessible to non-developers.

The HbbTV applications generated using the editor use a simple, menu-based structure, which can include different types of subpages, for example, lists, text pages, polls, prize draws, video clips, and even a Twitter overlay. The main customizable parts include text elements, images, colours, and available menu items. The editor also allows the user to easily preview the HbbTV application in the browser at any time before the eventual deployment to television.

With the increasing popularity of smart televisions and virtually ubiquitous HbbTV support in new receivers, there is great demand for HbbTV applications both in Finland and abroad. A product which can be used to quickly generate new HbbTV applications would certainly find its place in this environment, be it as a product for customers or as a tool for internal development.

2 HBBTV

2.1 The HbbTV standard

A smart television, or smart TV, is a television that can connect to the internet (Better Software Group 2021). Smart television applications may come pre-installed or be downloaded from an application store, and there are many different types, for example, applications for streaming services, catch-up TV, or social networking sites (Pratt 2022).

Meanwhile, HbbTV stands for Hybrid Broadcast Broadband TV, and it is a technology that combines a traditional DVB-standard television broadcast with broadband, connecting to both networks in parallel. The application data is generally transmitted via the internet, but the technology also allows data to be transmitted through the broadcast signal using a so-called DSM-CC object carousel. (Fallahi, Joki & Teirikangas 2019, 22.) Unlike traditional smart television applications, HbbTV applications do not have to be installed on the device. For the end user to be able to access HbbTV services, the broadcaster needs to offer the service, and the user needs a HbbTV-capable receiver with the service enabled as well as an internet connection (Fischer 2020, 906–907).

One usage of the HbbTV technique is creating an interactive web application layer on top of the traditional broadcast signal (Hasselström 2018), which is also the case in this thesis project. Typically, the HbbTV content will be invisible by default and can be accessed by pressing one of the colour buttons – usually the red button – on the remote control. Some HbbTV services are always available, including things such as video on demand (VOD) services and episode guides, while others are tied to the currently broadcasted programme, providing interactive programme-specific content. (Fallahi, Joki & Teirikangas 2019, 11-12.)

While smart television applications are somewhat platform-specific, with different manufacturers requiring different implementation solutions, HbbTV applications are standardized (Malhotra 2013, 12). The HbbTV specification is maintained by the HbbTV Association. The first version was published by the Association (then

called the HbbTV Consortium) and approved by the European Telecommunications Standards Institute (ETSI) in July 2010 (HbbTV n.d.-a). The latest version of the specification at the time of writing this is HbbTV 2.0.3, published in 2020 (HbbTV n.d.-c). However, older television models will still utilize older versions of the specification (Hasselström 2018). In Finland, HbbTV applications generally need to support HbbTV version 1.5 (Digita n.d.).

HbbTV is currently only available in certain countries, mainly in Europe, Oceania, and some parts of Asia and Africa (HbbTV n.d.-b). Other countries use different technologies (TM Broadcast 2020). However, HbbTV is still a significant market, having been implemented in 27 countries and reaching nearly 587 million people worldwide in 2018 (Fallahi, Joki & Teirikangas 2019, 7).

In Finland, HbbTV has been in use since 2012. There had been previous attempts to adopt the earlier MHP (Multimedia Home Platform) standard, which supported Java-based interactive applications. However, this technology failed in Finland due to a multitude of issues, and HbbTV later became the de facto interactive television standard. (Joki 2022; Malhotra 2013, 14.) In October 2021, HbbTV had 74% household penetration with 400,000 devices daily and 700,000 devices monthly connected to HbbTV services (HbbTV n.d.-b). Virtually all new televisions currently sold in Finland are HbbTV compatible, and programme-specific applications have a high participation rate, even up to 60% (Joki 2022).

2.2 HbbTV application development

HbbTV applications are largely based on the traditional web development technologies of HTML, JavaScript, and CSS, though some television and HbbTV-specific features must also be considered. For one, many television browsers will often not support the latest technologies, and older devices in particular are vulnerable to this. Thus, in order to support a wider range of devices, the use of ECMAScript 5 is recommended, and some newer CSS and HTML features may also have to be avoided. (Fallahi, Joki & Teirikangas 2019, 36–37.) To help developers, the HbbTV Association maintains a developer portal with examples, guidelines, and other resources for programming HbbTV applications (HbbTV n.d.-d).

Regardless of the strict compatibility requirements, some newer web technologies can still be utilized with the help of polyfills, transpiling, and monkey patching, which help translate the code into a format compatible with older devices (Hasselström 2018). This enables the use of more modern JavaScript frameworks, such as React or Vue. For example, Vue uses Babel to automatically manage transpiling and polyfills according to user-defined browser support requirements (Vue CLI 2022).

However, due to the immense diversity of television receivers of different brands and models, compatibility issues still often surface. Browser differences often cause different television models to behave vastly differently, and testing and error logging are also more difficult due to the browser console usually being inaccessible.

Furthermore, all applications for television have to consider the fact that most televisions have no keyboard and mouse and rely entirely on the remote control for navigation. Modern television remotes have certain standard buttons, such as the volume and channel keys, number keys, arrow keys (also called the directional pad), an OK button for accepting selections, and four colour buttons: red, green, yellow, and blue. Listeners for these can be added in the application, and they can be used for navigation. Because there is no mouse pointer, visually highlighting the currently focused item is particularly important. This is called focus management, and it is often implemented by, for example, altering the borders or colours of the focused item. (Langendijk 2021.)

One aspect where HbbTV development is simpler than smart television development is the resolution. While smart television applications have to support multiple different resolutions, HbbTV applications only need one: 1280 pixels horizontally by 720 pixels vertically (Mautilus 2018). This simplifies development, as many user interface elements can be configured using absolute values. However, it should be noted that the “safe area” – the area which can be trusted to be visible on all televisions – may be slightly smaller than this (Mautilus 2018), so some empty space should be left near the borders to account for this.

3 PROJECT BACKGROUND

3.1 Commissioning company

Sofia Digital is a Finnish company based in Tampere that provides services and solutions for televisions and other interactive devices, including developing software for smart televisions and HbbTV. The company was founded in the year 2000 and has clients and projects both in Finland and all around the world. The project was developed while working at the company, utilizing their existing solutions and know-how as well as their extensive testing zoo of over 130 television receivers. The graphics used in the example configuration of the HbbTV application were also provided by the company and are used with their permission.

3.2 Project introduction

The project started as a simple content editor paired with a single HbbTV application. Similar editors were already in use in older HbbTV projects, where each application had its own editor which could be used to customize its contents, such as text and images, without touching the source code. However, the technologies used in the older implementations were somewhat dated, using mostly PHP for the backend and plain JavaScript for the frontend. Furthermore, the editor design was rather basic, both in functionality and user experience.

This project was initially planned as an update to this concept: a more modern and controlled template project upon which future projects could be built. During the development, as more features were added to the editor, it was suggested that a single editor could instead control multiple HbbTV applications. This would improve the maintainability of the concept by reducing project fragmentation. From this idea, the project evolved into its current form: a general-purpose editor for generating HbbTV content.

With this editor, the customer will have more control over the content of their HbbTV application. The user will be able to generate and customize menu-based

applications with text pages, lists, submenus, and, by integrating with pre-existing Sofia solutions, user polls, prize draws, video clips, and even a Twitter feed. Similar applications have been used in the past as companion applications for television programmes for audience engagement and advertising. For example, a reality competition may have a companion application which is available during the broadcast of the programme and can be used to view more information about the participants or vote for one's favourite. The exact features and functionality of the editor will be described in more detail in chapter 4.

The main advantage of the editor concept is that changes can be made on the fly after the application has already been published. Compared to the previous implementations, this solution also provides more customization to the user, including custom fonts, application language, page-specific themes, and more detailed colour options. The new editor should also be easier to navigate for users with no technical background. On the developer side, giving more control to the customer should free up time for other tasks, and having the same editor control multiple applications will reduce the amount of duplicate work needing to be done for each project.

4 IMPLEMENTATION

4.1 Project structure and technologies

The thesis project consists of three applications split into two projects: the web editor and the HbbTV application. The web editor has a frontend, which is the user interface used to generate configurations for the HbbTV application, and a backend, which is an API that saves the configurations to a server and serves them onwards to the HbbTV application. The HbbTV application only has a frontend, which uses the editor configuration to generate its contents.

The project also utilizes other services to complement its functionality. These are mostly pre-existing services that were not implemented as part of the thesis project. They were mainly used to avoid having to rewrite critical functionality and to provide an extra layer of data security. For example, poll generation and vote counting are handled by an external application. Polls can be enabled, disabled, and further customized through the web editor, but the poll must be created, activated, and the voting options created through the poll application. The same application also serves the Twitter feed widget for the Twitter integration, and the feed can be customized through it. A separate backend layer, later referred to as the auxiliary backend project, is also utilized for certain functionality. This layer handles validating the voting requests, for example, counting whether the user has already used their allowed votes, collecting prize draw participation information, and serving data more securely and efficiently from the server to the end user. It also provides stable database storage for critical application data.

The technologies chosen for the project were Vue.js for the web editor frontend and the HbbTV application due to its lightweight nature, and for the API, the Node framework Express.js was chosen for its popularity and ease of use. Vue 2 was chosen over the more recent Vue 3 due to compatibility issues with some older television models.

For the editor, the Vuetify UI framework was used. Vuetify uses Google's Material Design components and speeds up development by removing the necessity to

design and build one's own components. Other frameworks were also considered, but Vuetify was ultimately chosen for its popularity, good documentation, and wide selection of components. For the HbbTV application, no UI framework was used to minimize compatibility issues with different television models.

The code formatter Prettier and code analysis library ESLint were used in all parts of the project to ensure code quality. The Vuex library was used for global state management.

There was very little code reuse from the previous applications, aside from the aforementioned pre-existing services. Of these, the auxiliary backend was not utilized directly but cloned from a previous implementation as a separate project to allow for major changes without interfering with any existing applications. The auxiliary backend project uses Express.js for the API, Redis data store for caching, and the relational database MySQL for storage. While some functionality of the auxiliary backend project was also altered by the author, it should not be considered directly part of the thesis project but simply an interface used by the project applications.

The deployment to different environments (internal, staging, production) is handled through the automation server Jenkins, but the deployment details are also out of scope for this project.

4.2 Web editor backend

The editor backend is a simple Express application whose main function is saving and serving the editor configurations in simple JSON format. The configurations may also include other assets, such as images or fonts, which are also stored and served by the editor backend. This API will be protected in the live version of the application, and only certain parts of it will be publicly accessible via the auxiliary backend.

All data on the server is organized by the application (or user) name. All calls to the API must provide the user name as part of the query in order to access the

relevant data. The configurations are organized into the following categories: live configuration, live backup, named configurations, and temporary configuration. The live configuration is the one shown to the end user of the HbbTV application, the live backup is the copy of the previous live configuration, named configurations are saved configurations not shown to the HbbTV end user, and the temporary configuration is the configuration used to generate the preview of the HbbTV application for the web editor user.

The API endpoints are fairly straightforward. The caller can get a list of all configurations by the given user. This is a sorted list that includes the live configuration, live backup, and all the named configurations sorted by creation time, but not the temporary configuration. The user can also ask for a specific configuration by name.

Named configurations can be saved by sending the configuration data along with the desired name to the API. Only the ten newest named configurations are kept, so calling this endpoint removes the oldest one if the maximum number is exceeded. Temporary configurations have their own endpoint, as only one temporary configuration can exist at any given time.

The live configuration and live backup work slightly differently. The auxiliary backend is ultimately responsible for storing and serving the live configuration, so the endpoint for retrieving the live configuration fetches it from the auxiliary backend. Upon saving a new live configuration, the call is also forwarded to the auxiliary backend. A local server copy is also created as a backup. When a new live configuration is created, the previous configuration is saved as the new live backup. There is currently no endpoint for deleting configurations, as the deletion of old configurations is handled automatically.

The images and fonts used by the application also have their own endpoints through which they can be retrieved, uploaded, or deleted. Multiple images can also be uploaded simultaneously, which was implemented using the Multer middleware.

4.3 Web editor frontend

4.3.1 Overview

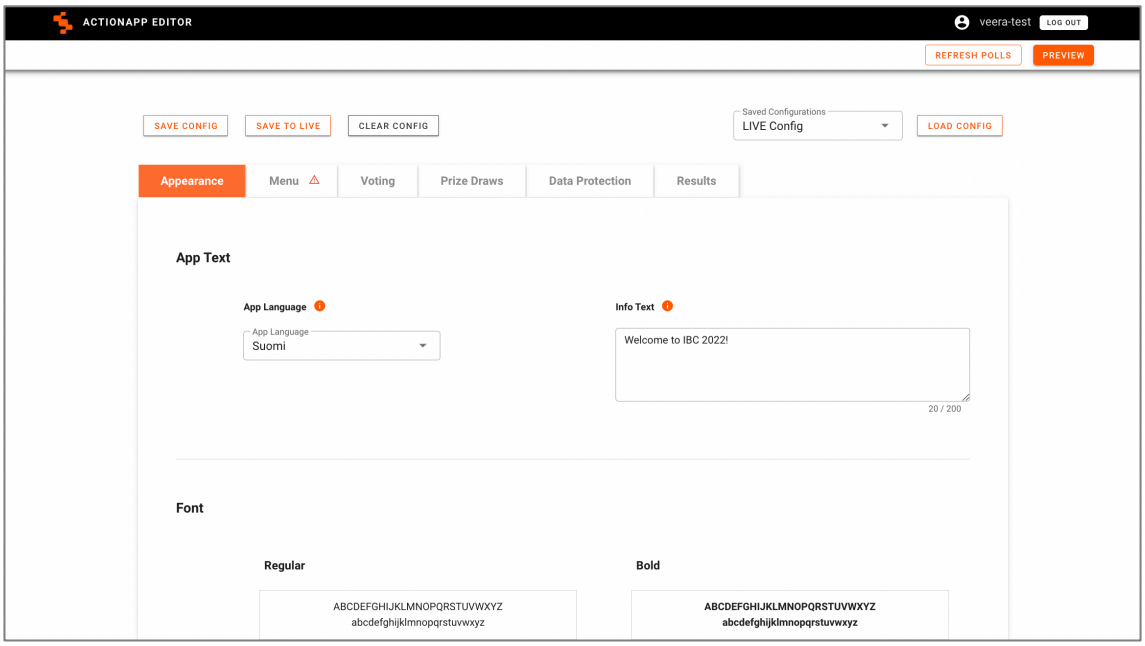
The editor is perhaps the largest section of the project. In addition to the Vuetify UI framework mentioned earlier, Vue Router is used to manage the different application views, and the Vuex library is used for global state management. State management is particularly important because in order to remain consistent, all parts of the application must be aware what the current user-defined configuration is at all times. Every time the user makes a change in the editor, it is immediately saved in the global state so that it can easily be used to generate a new configuration when the user makes a save or opens the application preview. If the state is not manually saved, the changes are lost. To prevent accidental data loss due to this, the user is warned on closing or reloading the editor page that their changes may not have been saved. This feature is only active in the deployed application, as otherwise it would prevent Vue's hot reload and slow down development.

When opened for the first time, the editor opens to a simple login screen (PICTURE 1). At this stage of development, only a valid user name is required to log in. The user name is valid if it corresponds with an active user of the separate poll application. The browser's local storage is used to remember the logged in user, so the page can be reloaded or closed and reopened without having to log in again. A better and more secure login system will be implemented before the application is formally released.

PICTURE 1. Editor login dialog.

Once logged in, the user will see their user name on the right side of the floating top bar of the app, along with a small logout button. There are six views in the

editor, accessed by tabs: Appearance, Menu, Voting, Prize Draws, Data Protection, and Results. The first tab will be selected upon login (PICTURE 2). If a live configuration is found for the user, it is automatically loaded. Otherwise, an empty default configuration is used.

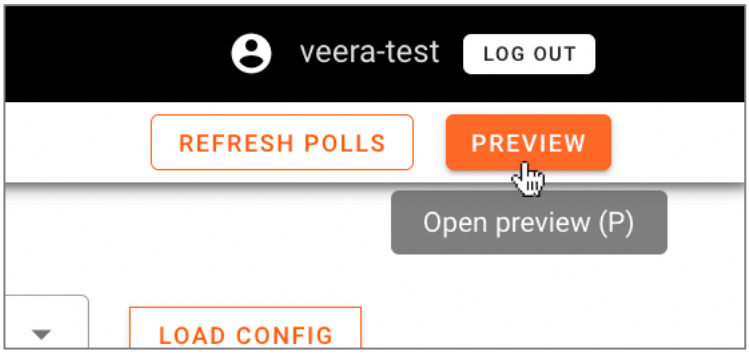


PICTURE 2. Front page of the editor.

Above the page navigation are the buttons for managing configurations. The “Save Config” button saves the current state of the editor as a new named configuration. On clicking it, a dialogue prompt opens asking for the name for the configuration. The “Save to Live” button saves the current configuration as the live configuration – that is, the configuration used in the HbbTV application shown to the television viewers at home. The “Clear Config” button loads a default state of the editor with empty fields and no menus but does not affect any of the saved configurations. The “Load Config” button on the right loads the selected configuration from the list of configurations selected in the dropdown. This list is fetched from the API.

All the save and load operations are Vuex store functions which call the API when appropriate and convert between the JSON format utilized by the API and the store state, which is a JavaScript object.

There are two more buttons in the top bar: Refresh Polls and Preview (PICTURE 3). The Refresh Polls button simply re-fetches the user data from the poll application which was loaded on login. The uses of this data will be explained further in the Menu and Voting sections of this chapter. The Preview button opens the HbbTV application in a new browser tab. The application will be generated based on the current editor configuration, regardless of whether it has been saved or not. The hotkey P will also open the preview if no other elements that use the keyboard, such as text fields, are currently active.

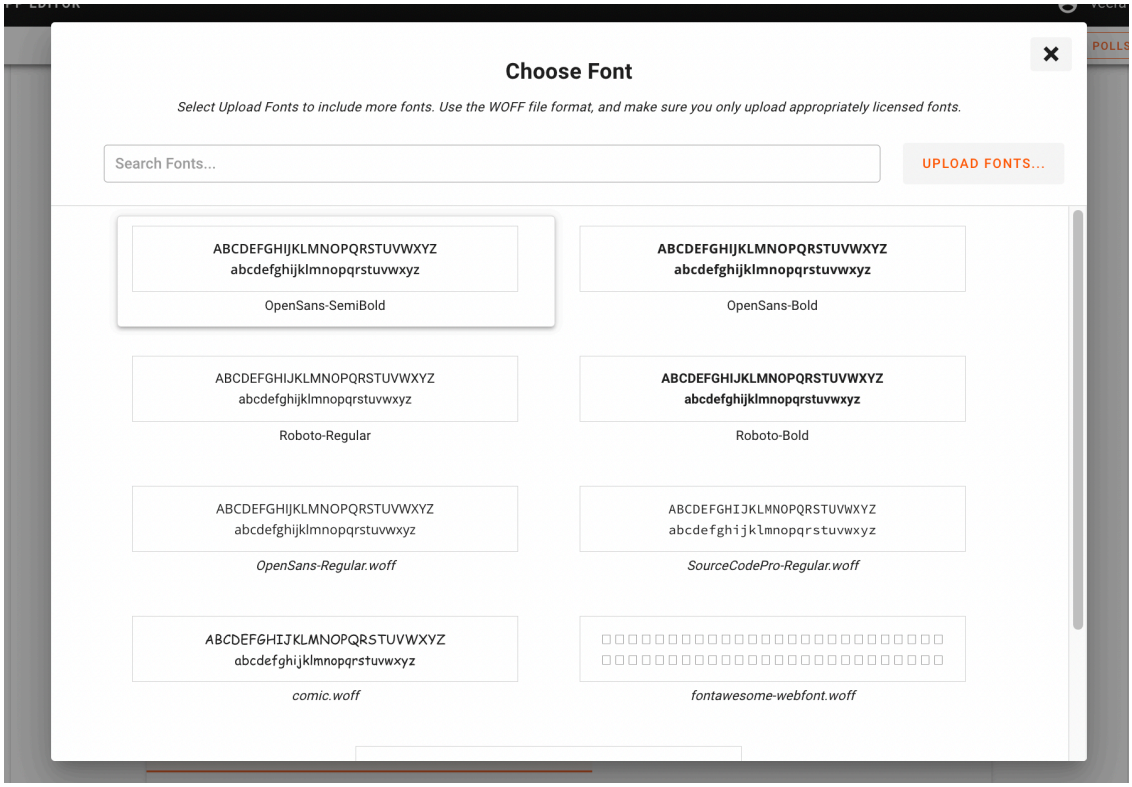


PICTURE 3. Top bar buttons.

4.3.2 Appearance options

The Appearance tab contains options to customize certain global elements. For example, the application language selection controls the language of certain non-customizable elements in the HbbTV target application, such as dialogue options and button labels. The only languages currently available are Finnish and English.

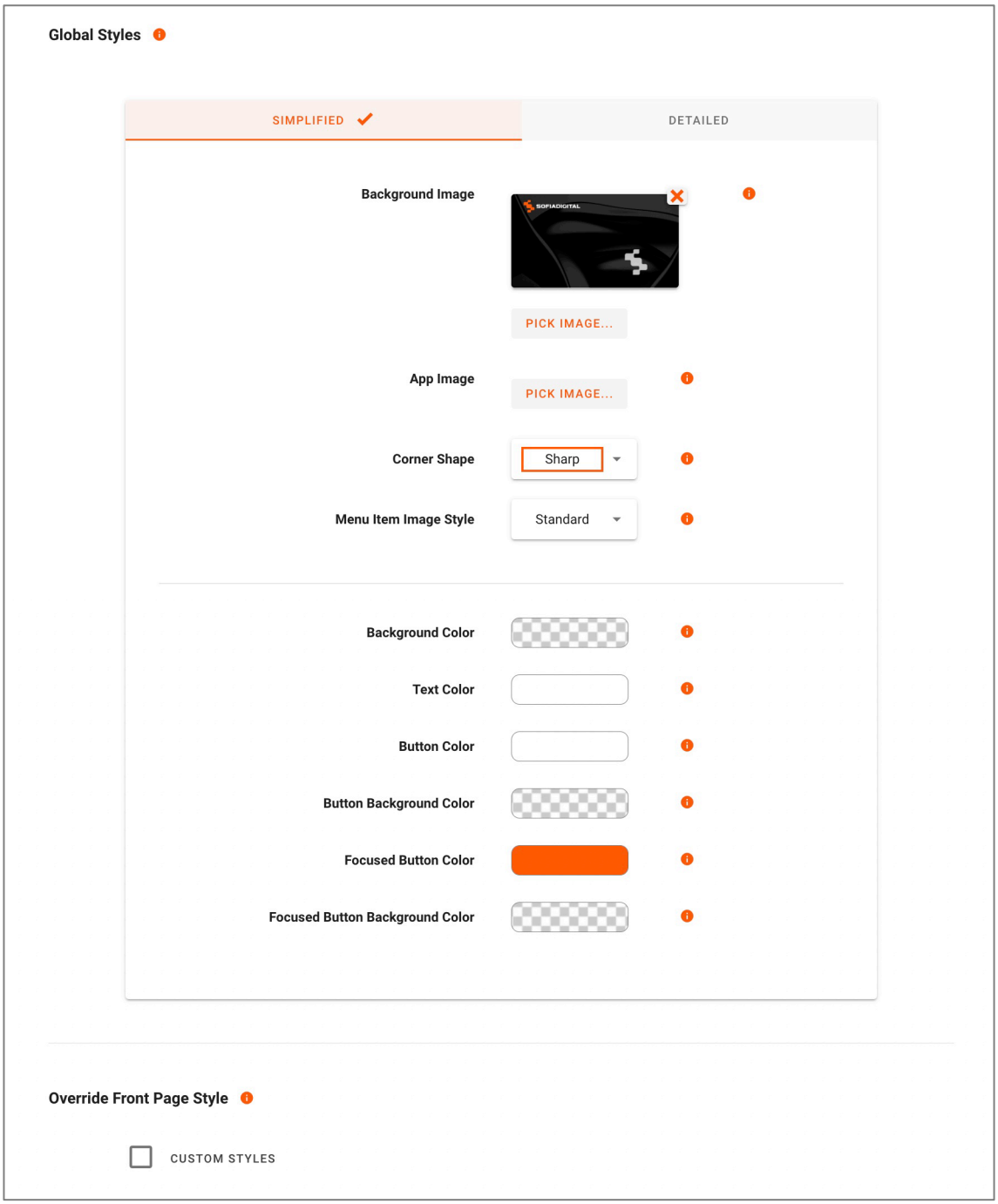
The application font may also be changed here using the font picker dialogue (PICTURE 4). There is a small number of default fonts available, but the user can also upload and use their own fonts. However, only the WOFF (Web Open Font Format) file format is supported due to HbbTV limitations, and the user is also warned to be aware of possible licensing limitations.



PICTURE 4. Font picker dialogue.

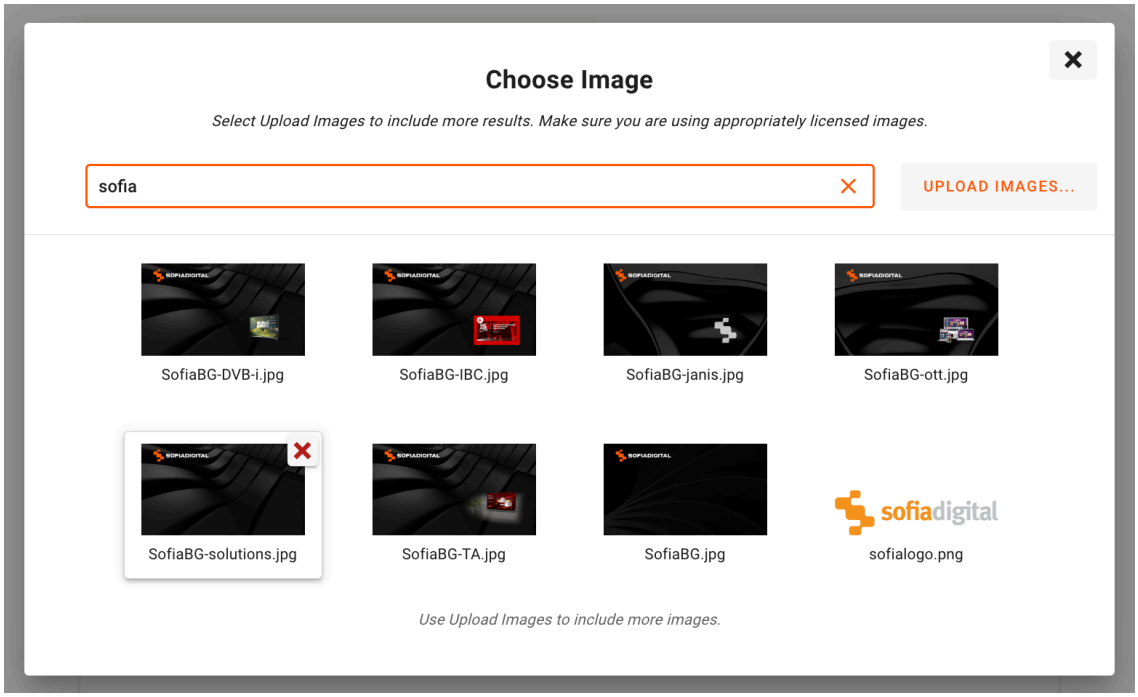
The Appearance page also has a styles component (PICTURE 5), which allows the user to customize other application styles, such as the application background picture, menu item style and shape, and the colours of different elements. The styles component has two tabs: the simplified tab has fewer options and one selector controls multiple styles, whereas the detailed tab allows the user to customize each element separately. The small “i" button next to each field gives more information regarding which fields will be affected by the selection. The detailed tab has 32 options in total, which is why the simplified tab was added to serve as a less overwhelming starting point.

Under the Global Styles component, there is a checkbox labelled “Override Front Page Style”, which, when enabled, opens a similar component for selecting different styles for the front page of the application. For example, the user may wish to have a front page with a different background image and colours than the rest of the application. There are also similar selectors for each subpage, found in the subpage options under the Menu tab. The current global styles will always be the starting point for these style overrides.



PICTURE 5. Styles component.

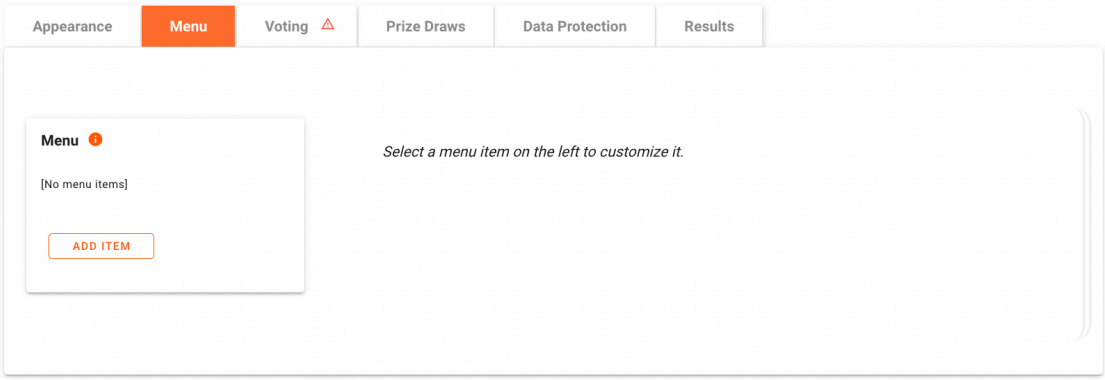
The style component also uses the image picker (PICTURE 6), which is similar to the font picker presented earlier. The user can search, upload, and delete images and then select the one they wish to use in the application. Multiple images can also be uploaded at the same time, allowing the user to quickly upload large collections of image assets.



PICTURE 6. Image picker dialogue.

4.3.3 Menu configuration

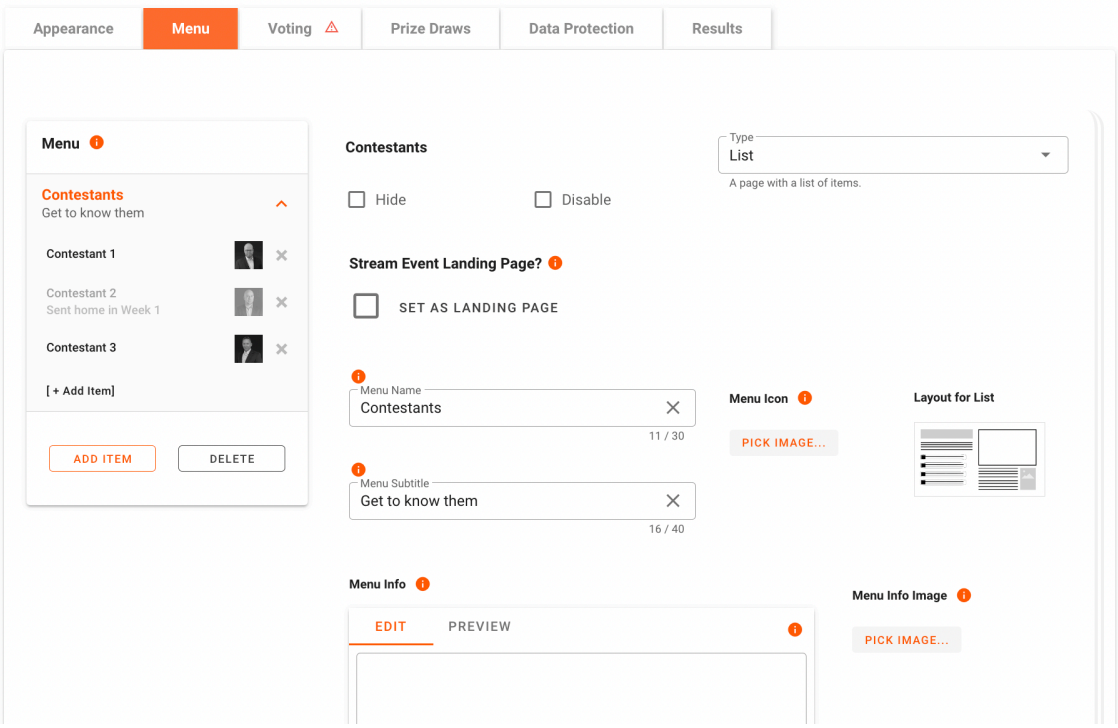
The menu configuration is perhaps the most complex part of the editor. An empty menu tab with no menu items can be seen in PICTURE 7. New menu items can be added using the “Add Item” button in the menu preview on the left, and the menu item settings will open on the right side of the page. Menu items can be reordered at any time simply by dragging them, which was implemented using the “vuedraggable” library.



PICTURE 7. Empty menu tab.

There are several different menu item types. By default, new menu items are created as the list type. A list is a menu page with one or more list items. These

list items look identical to the main menu items but do not lead to any subpages. When a list item is focused, information about the item will be visible on the right side of the HbbTV application. This type of menu item can be used to provide different kinds of information, for example, listing all the contestants in a reality television series and their descriptions. An example of a list page being configured can be seen in PICTURE 8.

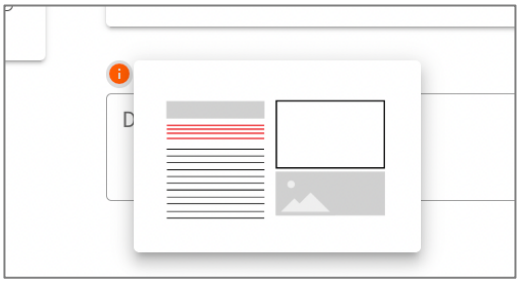


PICTURE 8. Menu item configuration for the list type.

The menu item type can be changed from the dropdown menu in the top right corner of the menu item settings. The general layout of the currently selected type can be seen as a small preview picture under the type selector.

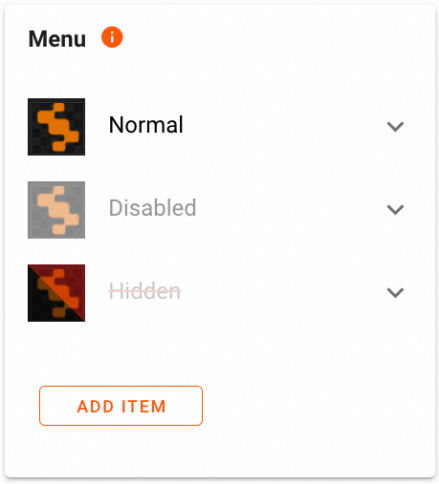
Each menu item can be selected on the left and configured using the fields that open on the right side of the page. Many of the configuration options are the same between the different menu item types, but some fields change depending on the type. Some text fields have soft character limits which warn the user when the text length might exceed what will fit on the page. List and voting page items are configured similarly to the main menu items and have their own, more limited options.

To help the user understand what they need to do, the application utilizes hints quite extensively. Each configurable field has a small “i" icon next to it, and clicking it will reveal further information on the effects of the field. Some are text hints, while others show a small preview of the page layout with the selected field highlighted, as can be seen in PICTURE 9. Certain elements also have hover hints.



PICTURE 9. Layout hint for the page description field.

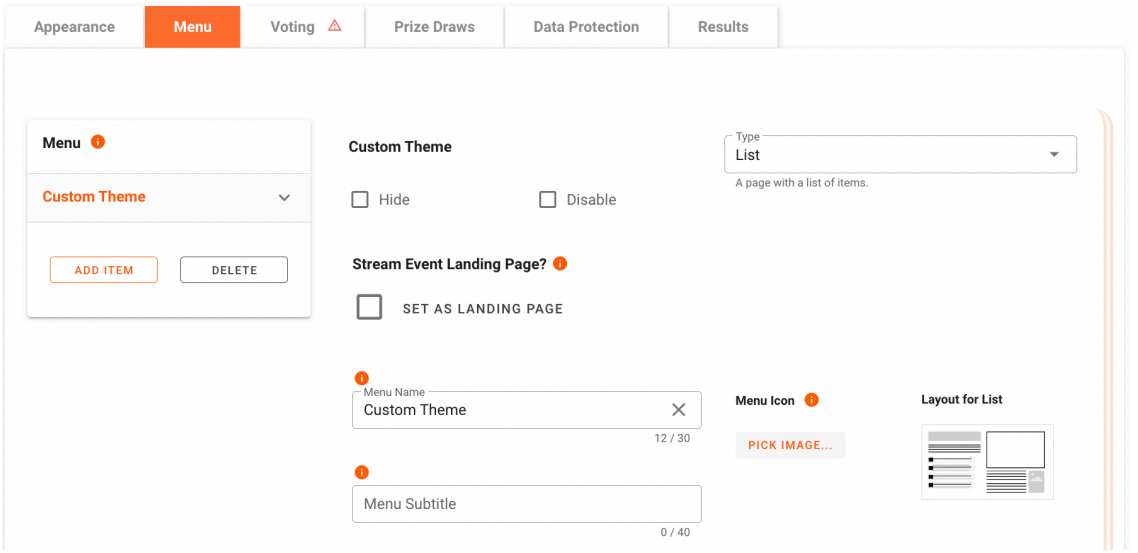
Each menu or list item can be given a title and a subtitle as well as a menu icon. All menu and list items can also be hidden or disabled. Hidden items are not shown in the target application, while disabling an item simply greys it out and disables any functionality, such as entering subpages or voting. The state of each item is also reflected visually in the menu preview, as demonstrated in PICTURE 10.



PICTURE 10. Menu item states.

Any menu item can also be configured to have its own theme separate from the global theme, as explained in the previous section. This is done by checking the option labelled “Custom Styles” at the bottom of the options page, which reveals

the style picker. When the custom styles are enabled, the grey indicator lines on the right side of the options section light up so that the user can easily see when custom styling is in use (PICTURE 11).



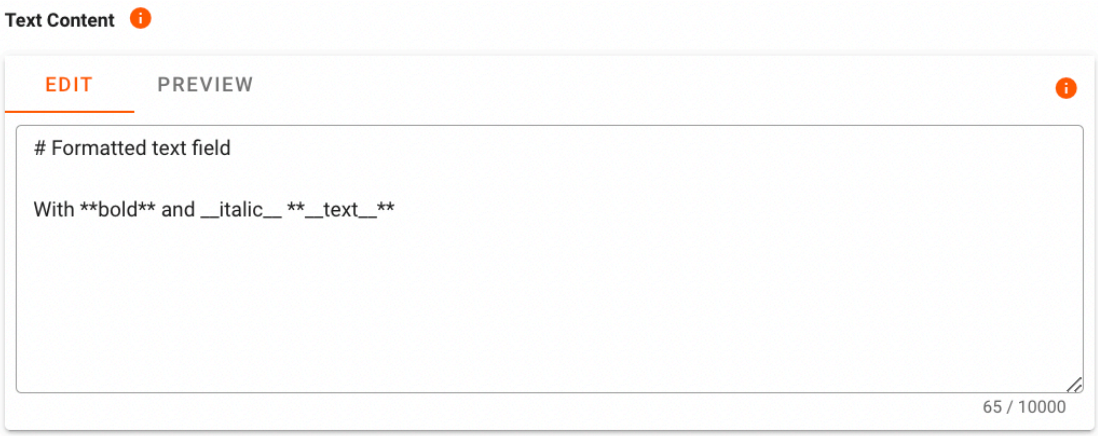
PICTURE 11. Page with custom styles enabled.

Any menu item can also be set as an alternative landing page for the application by selecting the checkbox labelled "Set as landing page". The landing page refers to a special route in the HbbTV application which can be used as the application landing page when opening the application. This route will automatically redirect to any page set as the landing page in the configuration. This editor option allows dynamically changing the landing page of a live application at any time. Naturally, only one landing page can be set at a time, so checking this box clears any previous selection. If no landing page is configured, the main page is used instead.

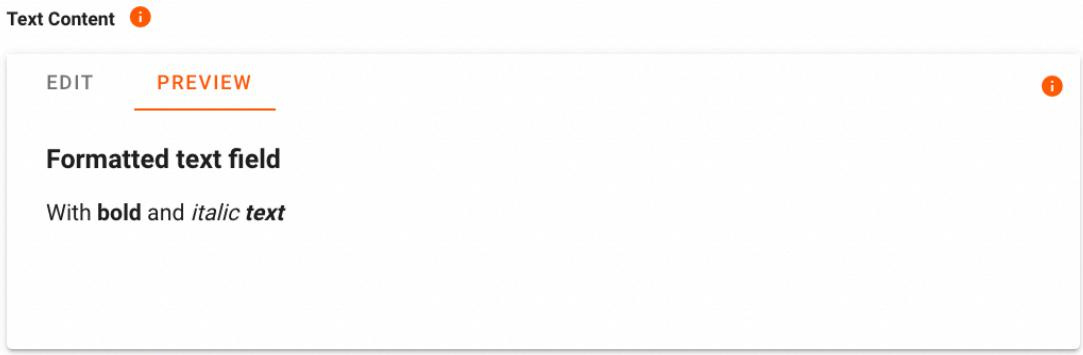
In addition to lists, other menu items include voting, prize draw, page, Twitter, clips, and submenu. The voting and prize draw types are slightly more complex, so they are discussed in more detail in their own separate sections.

A page is a simple text page with no additional functionality, which allows showing a larger amount of text to the user. The user can scroll the text up and down using the up and down keys on the remote control. Additionally, an image can be configured to be shown next to the text.

The main text configuration for the page type uses a special formatted text field, which allows certain text enrichments, namely headings, bold text, and italics. This is done using special syntax similar to the Markdown mark-up language, with bolded text being surrounded by two asterisks, italicized text by two underlines, and headings starting with a hash symbol at the beginning of a line followed by a space. The text box has an info button which explains the syntax to the user. More elaborate styling is not supported in order to limit any compatibility issues with different television models. The text box has two tabs, one for editing and one for previewing the text. PICTURE 12 shows the edit view with some formatted text, while PICTURE 13 shows the preview of that same text. The parsing of the formatting symbols is explained in more detail in the section describing the HbbTV application.



PICTURE 12. Formatted text field, edit view.



PICTURE 13. Formatted text field, preview tab.

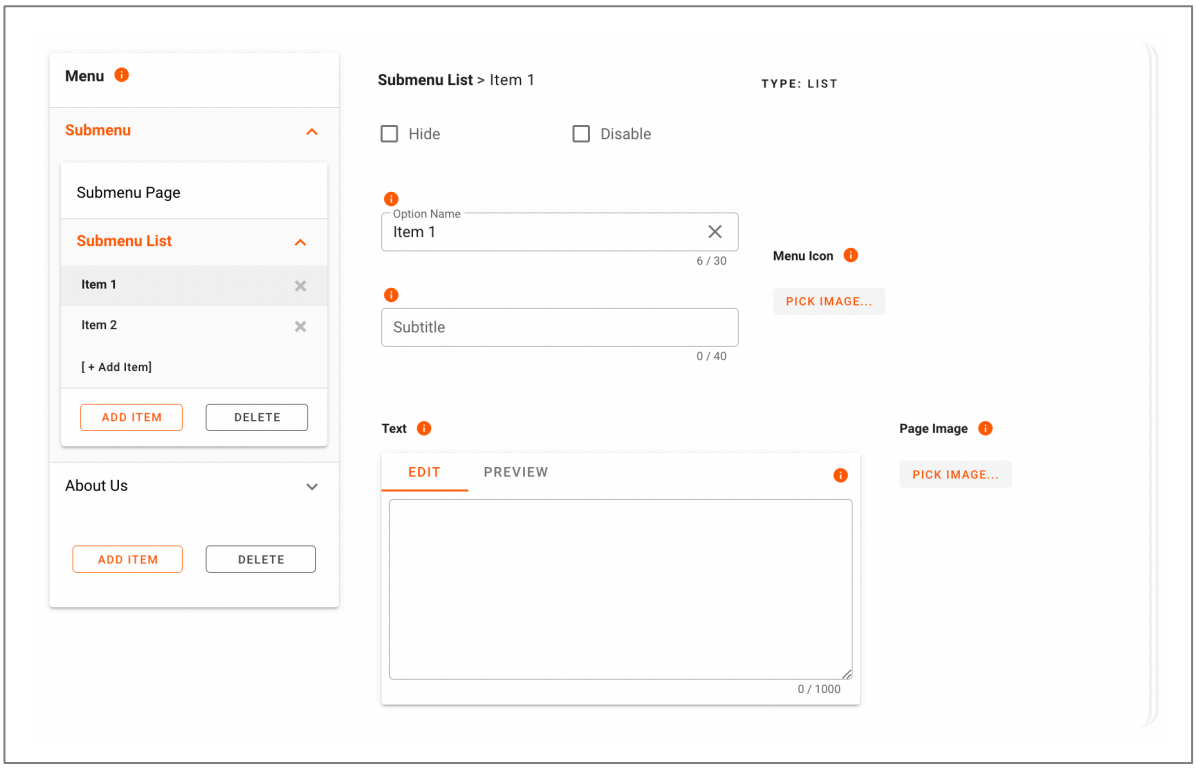
This formatted text field is also used in certain other fields in the editor where longer text is expected, such as the menu item description shown on the parent menu page when the menu item is focused.

The Twitter page type generates a page with a Twitter feed overlaid on top of the current programme. The Twitter feed is a separate widget created prior to this project, and its configuration must currently be done entirely through the poll backend, making its usage and implementation details out of scope for this project. The editor simply allows the user to add or remove it from the HbbTV application.

Only one Twitter feed can currently be configured for each application. There is also an automatically created colour button shortcut for the Twitter page in the HbbTV application. If the editor user only wants this shortcut without a corresponding menu item, they can create a Twitter page and simply hide it in the configuration. The HbbTV application will still detect it and create the colour button which opens the feed.

The clips menu type is a page where user-provided video clips related to the application can be selected and played. This page type is not yet fully implemented due to missing functionality in the auxiliary backend, but the main concept is that the editor user can set the clips page description in the application, and the rest of the menu and its functionality will be automatically built based on data retrieved from an API endpoint selected by the user, which needs to conform to a certain data structure.

The submenu page type creates a menu similar to the main level menu where other pages can be created as children (PICTURE 14). Submenus can also have other submenus as children, but the submenu depth is currently restricted to two levels for the sake of the usability of the HbbTV application. However, this restriction would be trivial to lift if needed. The submenu logic is implemented using simple recursion, both in the configuration hierarchy and the component structure, and the menu component tracks the depth of the recursion in order to limit the maximum depth.

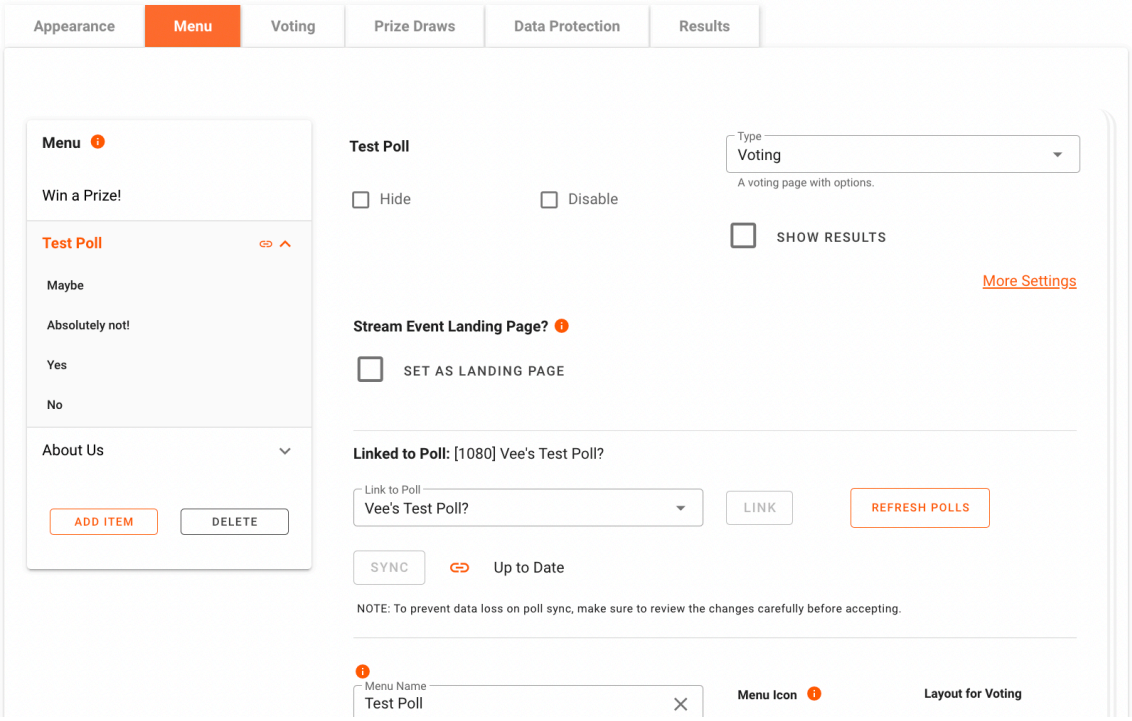


PICTURE 14. Submenu configuration.

When changing the type of a menu item, some data is automatically transferred to equivalent fields in the new type. Certain data, for example, poll linkage information when changing between a voting page and a list, is retained in the configuration even though it is not used in the new page type. This is so that the user can comfortably switch between menu types without losing too much data. Other data, such as submenu pages, are destroyed on changing away from that type in order to avoid unnecessarily increasing the size of the configuration file. The user is always warned via confirmation dialogue regarding destructive operations.

4.3.4 Polls and voting

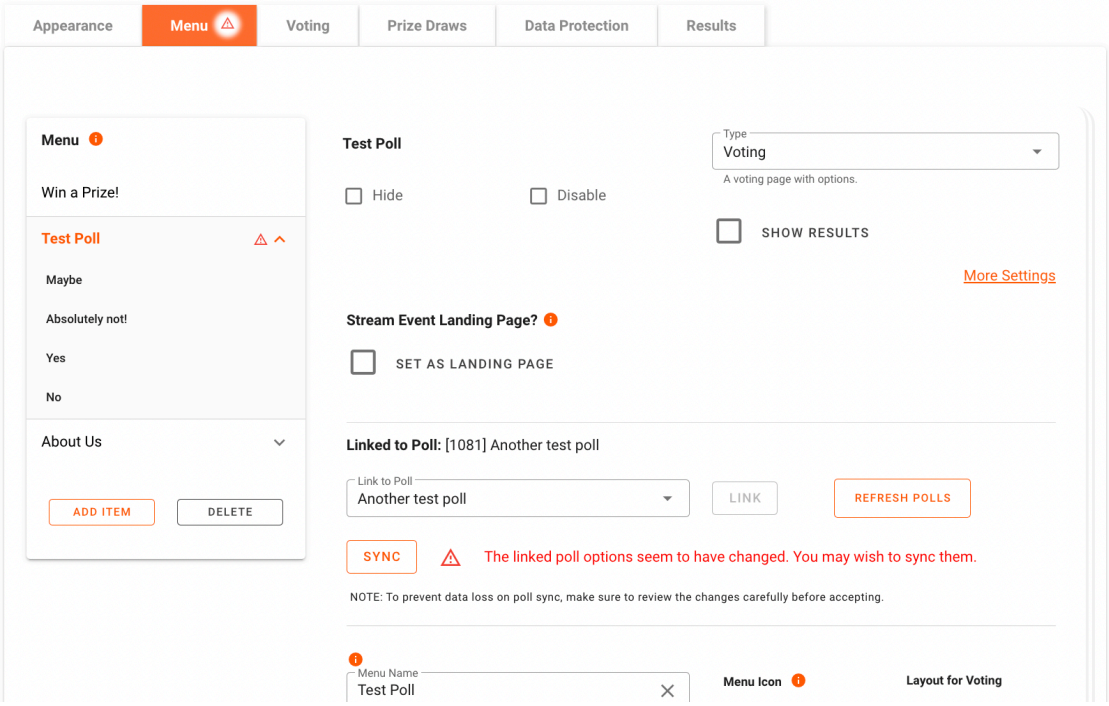
Configuring voting pages in the menu configuration works similarly to lists, with a few additional features which can be seen in PICTURE 15. Firstly, there is a checkbox labelled “Show Results”, which makes the current poll results visible on the voting page. This cannot be enabled by accident, as attempting to check the box triggers a confirmation dialogue warning the user that with this option, the poll results will be visible in the HbbTV application.



PICTURE 15. Voting page settings in the menu configuration.

Secondly, voting page menu items cannot be added and deleted manually. Each voting page much be synchronized to match a user poll in the poll application, and the menu item titles are determined by the voting options in the selected poll. This is to ensure that user votes will always be sent to the correct option in the correct poll and never inadvertently switched around due to accidental renaming.

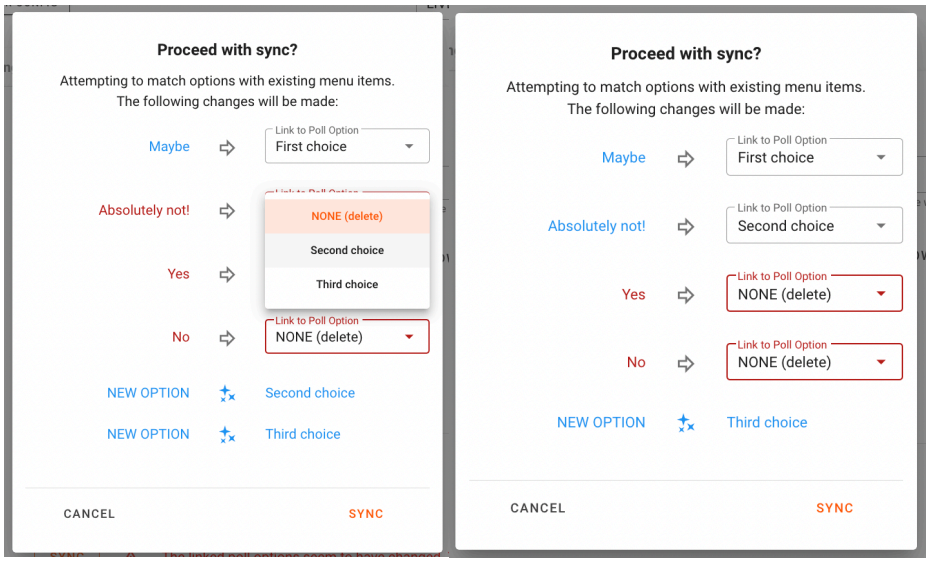
When the poll is synchronized, a link icon will be shown both on the menu item as well as next to the “Sync” button. On the other hand, when the voting page state does not match the selected poll options, a warning icon is shown instead, as can be seen in PICTURE 16. Another warning icon is shown on the Menu tab itself, indicating to the user regardless of where they are in the application that something on that page requires their attention.



PICTURE 16. Out-of-sync voting page in the menu configuration.

When a poll is out of sync, the user must first ensure that the correct poll is linked to the voting page using the poll dropdown menu, which lists all the polls available to the logged in user, as configured through the poll application. The “Refresh Polls” button can be used to manually retrieve new poll data from the poll application if changes have been made there after the user data was loaded into the editor.

Next, the user must synchronize the poll using the “Sync” button, which opens the pre-synchronization dialogue. This dialogue attempts to match any existing menu items to the poll options based on their titles. Items that cannot be matched will be deleted, and new options will be created as necessary. The user can also override the automatic pairings by selecting the matches manually from a dropdown menu, as demonstrated in PICTURE 17. The advantage of matching a poll option with a pre-existing menu item is that the menu item will retain all the data previously configured for it, such as text and images. After the user is happy with the selection, they can start the synchronization. When the synchronization is complete, a post-synchronization dialogue is shown, informing the user how many items were deleted, renamed, and/or added.

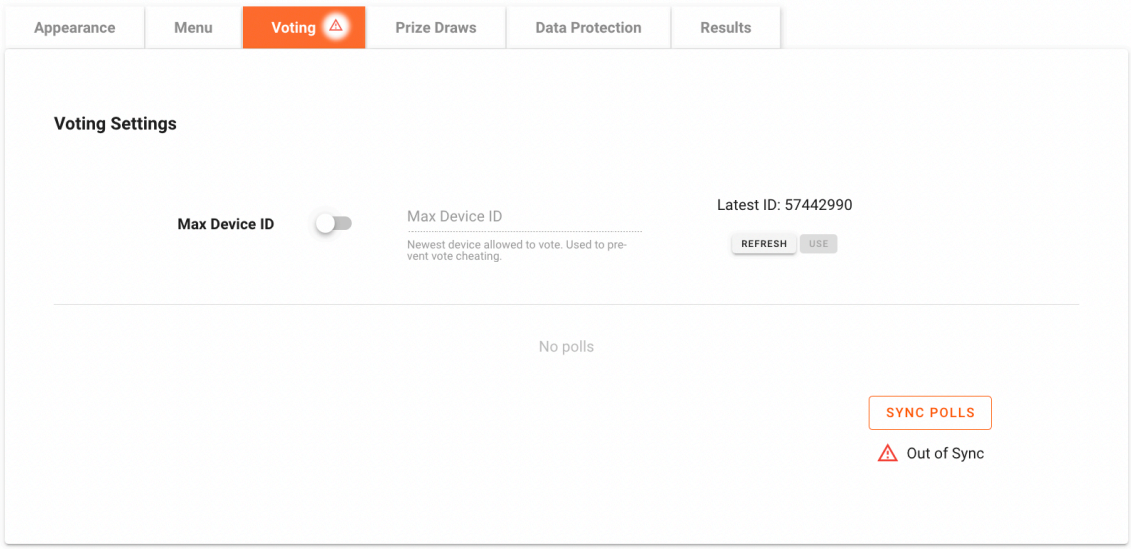


PICTURE 17. Pre-synchronization dialogue.

While the individual voting pages can be configured through the menu configuration, the separate “Voting” tab exists to provide general configuration options for each poll and to show all available polls on a single page. The “More Settings” link in the menu item settings also directs the user to this page. Here the user can find settings related to polling times and vote validation for each poll, in contrast to the more visual options in the menu configuration.

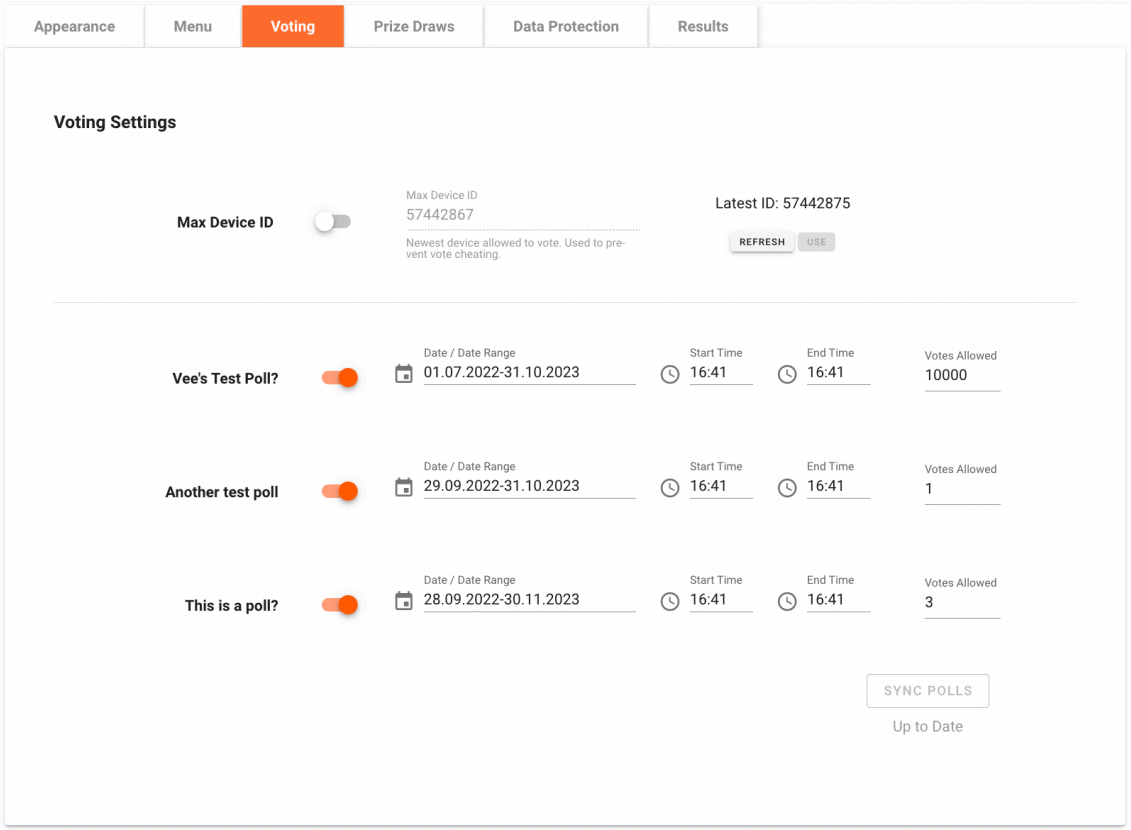
The overall polling period for each poll is configured in the poll application, but polls can also be temporarily enabled and disabled through the editor. This is to allow, for example, setting a certain weekly time window for voting without having to reset the poll each time.

In an empty configuration, no polls are initially shown on the list (PICTURE 18). The user has to manually select “Sync Polls” to pull the up-to-date list from the poll application. The editor also detects if the polls are not synchronized with the poll application data and shows warning indicators on the tab and under the synchronization button until all the polls are synchronized. Automatic synchronization is disabled to ensure the editor user always notices critical changes to poll data. For this purpose, a dialogue recapping the changes is also shown after each synchronization.



PICTURE 18. Out-of-sync Voting tab.

After synchronizing the polls, the user has to enable them and set a date and time range for the poll to remain active. Vuetify date and time pickers are used for these fields. The number of votes allowed per user in each poll can also be configured here. A synchronized view of the page can be seen in PICTURE 19.



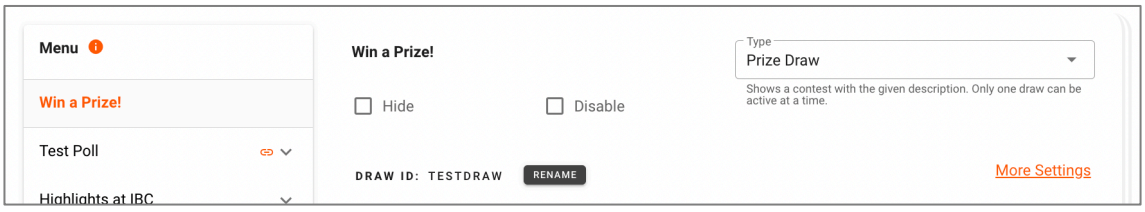
PICTURE 19. Synchronized Voting tab.

The “Max Device ID” field refers to an ID number assigned to every HbbTV end device in Finland. Without going into detail, using this field prevents vote cheating by resetting one’s device settings. The editor fetches the last assigned ID value from an external API, which can be assigned as the highest allowed ID for the polls. This solution has the side effect of also occasionally blocking innocent users with brand new televisions from voting, but this was deemed an acceptable trade-off for the added security.

4.3.5 Prize draws

When a new prize draw is created in the menu configuration, the editor asks the user for a draw ID, which is a unique identifier that will be used by the auxiliary backend to save the prize draw participation information. Two prize draw pages can have the same draw ID, but this means that they are ultimately the same prize draw, as all the participant information will be sent to the same location. The user is always warned when they are about to use a duplicate identifier. The input is also validated to only accept certain characters in order to make the ID database-compatible and prevent SQL injection attacks. The user is able to manually edit the ID after its creation, but this will unlink the draw from any pre-existing data in the database, so a warning will be shown instructing the user that the action may lead to data loss.

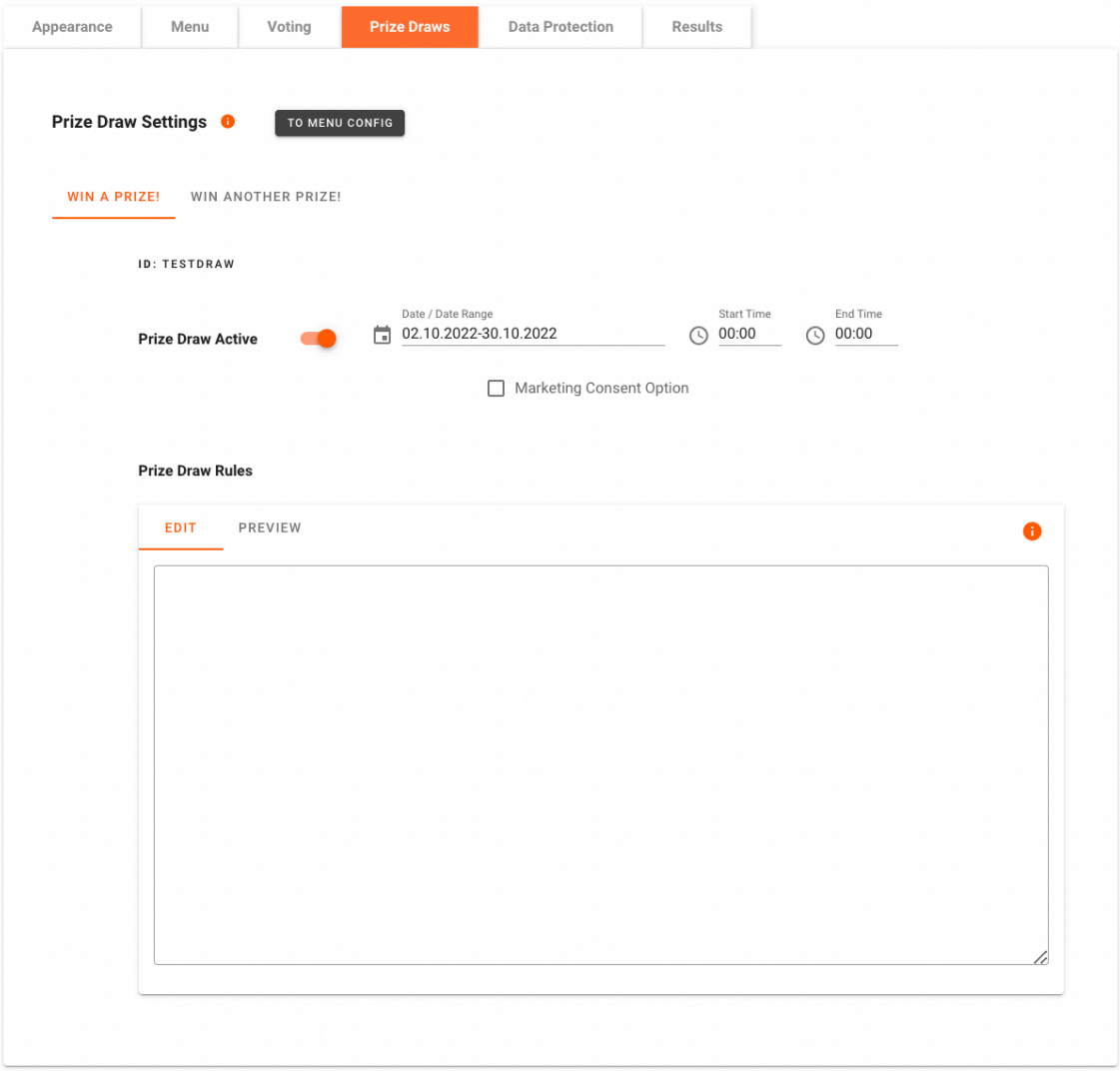
PICTURE 20 shows a part of the user interface for the prize draw page configuration. The “More Settings” link sends the user to the “Prize Draws” tab where more configuration options are available. On that page, there is also a return link which returns the user back to the menu configuration tab.



PICTURE 20. Prize draw settings.

The “Prize Draws” tab can be seen in PICTURE 21. Like the poll settings, the prize draw settings have been separated from the menu configuration to provide

the user easier access to all the existing prize draws in one view. All configured prize draws are shown as tabs which can be selected to edit the settings for each draw. The currently available settings are the time period when the prize draw should be active, whether to include a marketing consent option, and the prize draw rules, which uses the special formatted text field.

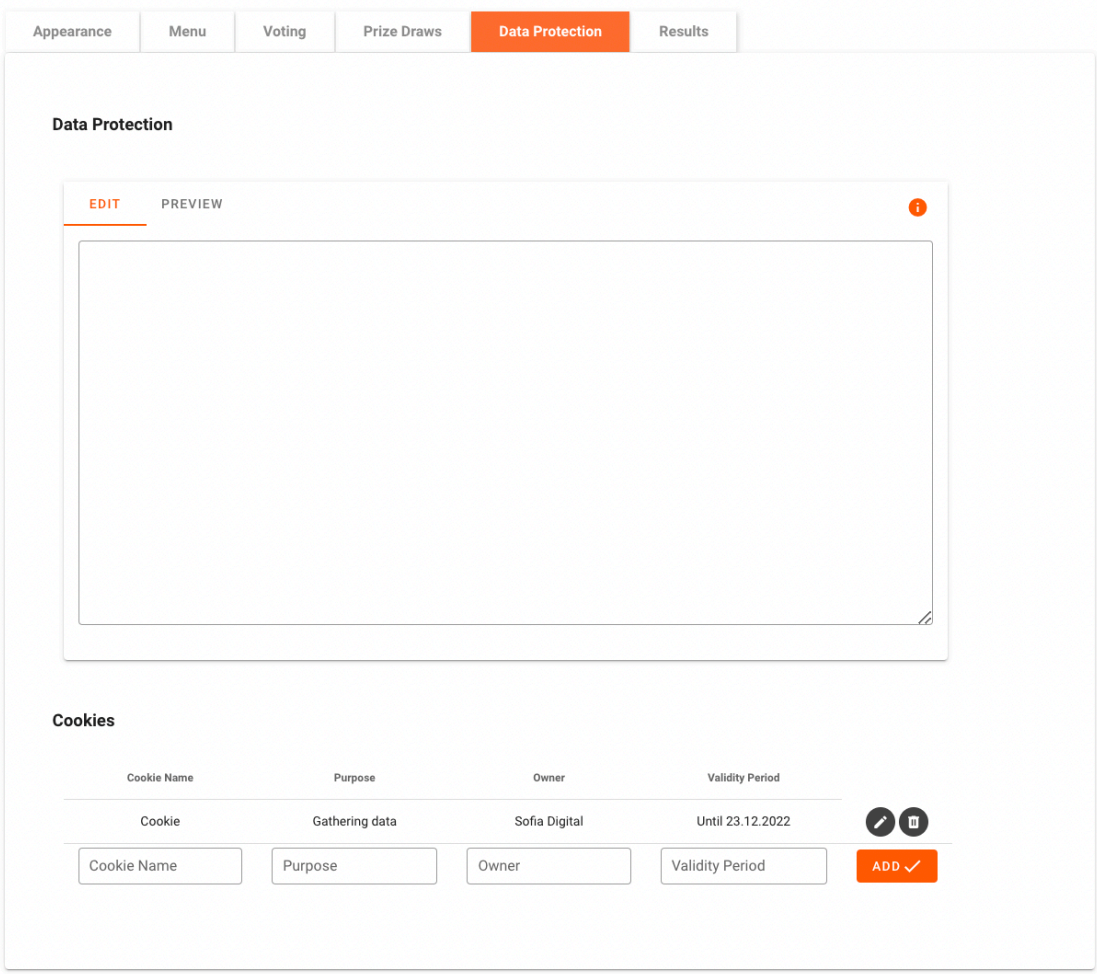


PICTURE 21. Prize Draws tab.

Selecting the marketing consent checkbox adds two participation options on the prize draw: to either only participate in the draw or to participate in the draw and also agree to receive marketing messages from the draw organizer. Application-wise, the only consequence is that the user's choice is received and saved by the auxiliary backend. Using the data will be the responsibility of the draw organizer.

4.3.6 Data protection

The data protection tab (PICTURE 22) allows the user to set the data protection text for the application. Formatting is available for this field, and there is no character limit. Cookie information can also be set using the dynamic list at the bottom of the view.

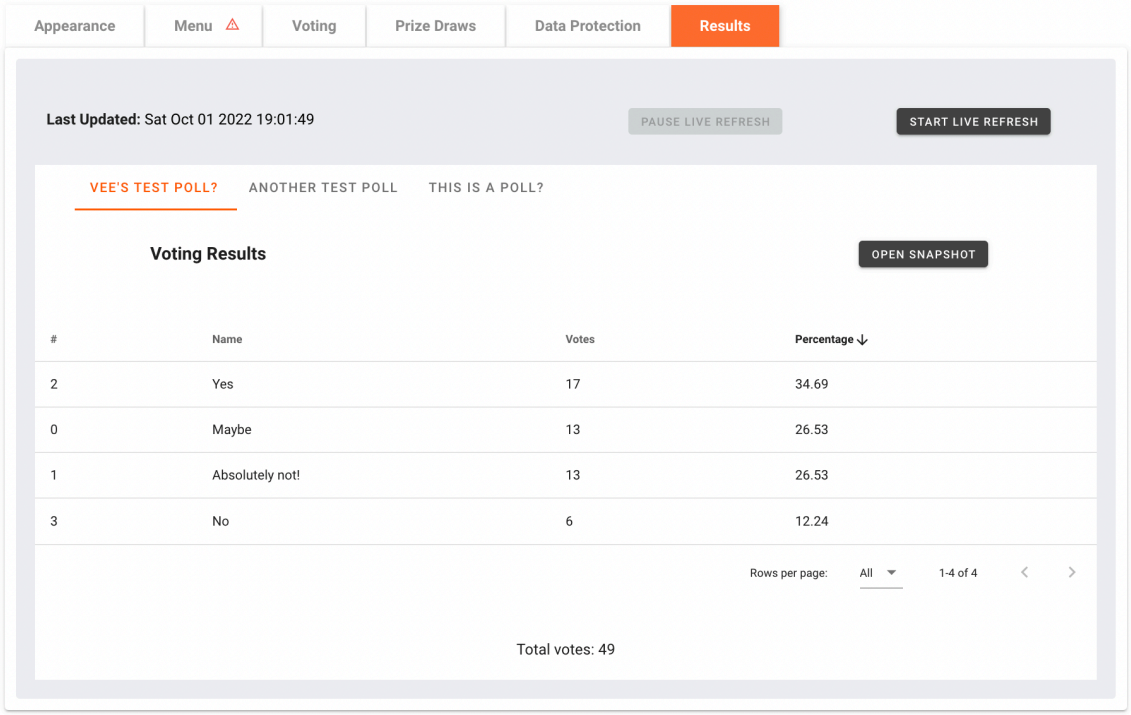


PICTURE 22. Data Protection tab.

4.3.7 Results

The results tab pulls the poll results from the poll application and presents them to the user in table format (PICTURE 23). Each poll is on its own tab, and the results can be ordered based on option index, name, or number of votes. A snapshot of the results can also be opened if the user wants to log the state of the poll

at a specific moment in time. This opens in a new window, which is created using simple dynamically generated HTML.



PICTURE 23. Results tab.

There is also an option to start or stop live refreshing the results. When enabled, the poll results are fetched and updated every five seconds. The live refresh is automatically paused on tab change to avoid unnecessary traffic to the poll API. The results are also updated on page reload or manually selecting “Refresh Polls”.

4.4 HbbTV application

4.4.1 Basic structure and routing

The HbbTV application was built on top of an internal company template project for Vue, which included all the necessary HbbTV application functionality, such as initializing the application as a HbbTV application, setting the screen dimensions, adding a special class for focusable elements, registering key mappings, listeners and navigation handlers for remote control key events, and initializing

the Vuex store. This chapter will focus on describing the functionality added to this base.

The main structure of the application relies on being provided the user or application name, which is required to access the correct configuration. The application simply shows an error message if a user name is not provided. The name can be provided as part of the application URL, from where it is read when the application is first loaded.

The application is launched slightly differently depending on whether the configuration to be loaded is the live configuration or a preview from the editor. The preview adds a special query parameter to the application URL, which tells the application which configuration to retrieve from the configuration API. This information is then saved to the local storage of the browser to keep track of the configuration in case the page is reloaded after the query parameters have been lost to route changes. If no query parameters are provided, the live configuration is loaded.

Even though the application to be loaded is determined by the user name, only the live configurations can easily be accessed by simply altering the URL. Other configuration types are protected using methods such as timestamping. As the live configurations are generally already published applications, this was not considered a privacy issue. If configurations need to be tested in a live-like environment prior to their publication, this can be done in Sofia Digital's internal testing environment, which is secure and cannot be accessed publicly.

The basic structure of the application also relies heavily on routing. The different page layouts are determined by routes. Each page type has its own route and corresponding view component in the application, and upon loading the configuration, the application pairs each configured page with a corresponding view using page keys generated from their position in the configuration. This means that the user could theoretically change the type of any page simply by altering the URL, but as the URLs are largely inaccessible to regular television users, this is not a major concern. Furthermore, in most cases, changing the page type should

not have any serious consequences beyond the amusement of the user, as features such as voting are protected with additional checks in the auxiliary backend.

Submenus are also generated using the same routing system. The hierarchy is stored in the page key, which is generated iteratively from the page keys of the parents of the submenu page and reverse engineered where necessary to match it again with the corresponding page in the configuration.

In addition to the page type routes, other routes include the home view, which is the top-level menu of the application, the landing page view, which searches for the landing page set in the configuration and immediately redirects the user there, and the console log view, which is used to log error messages on television end devices during testing and can be accessed by enabling the separate testing mode using an environment variable, after which it can be opened using the yellow button.

4.4.2 Navigation

The basic navigation in the application happens using the arrow keys to move the focus and the OK button to select an option. The back key is mainly used for cancelling dialogues and returning to the previous page.

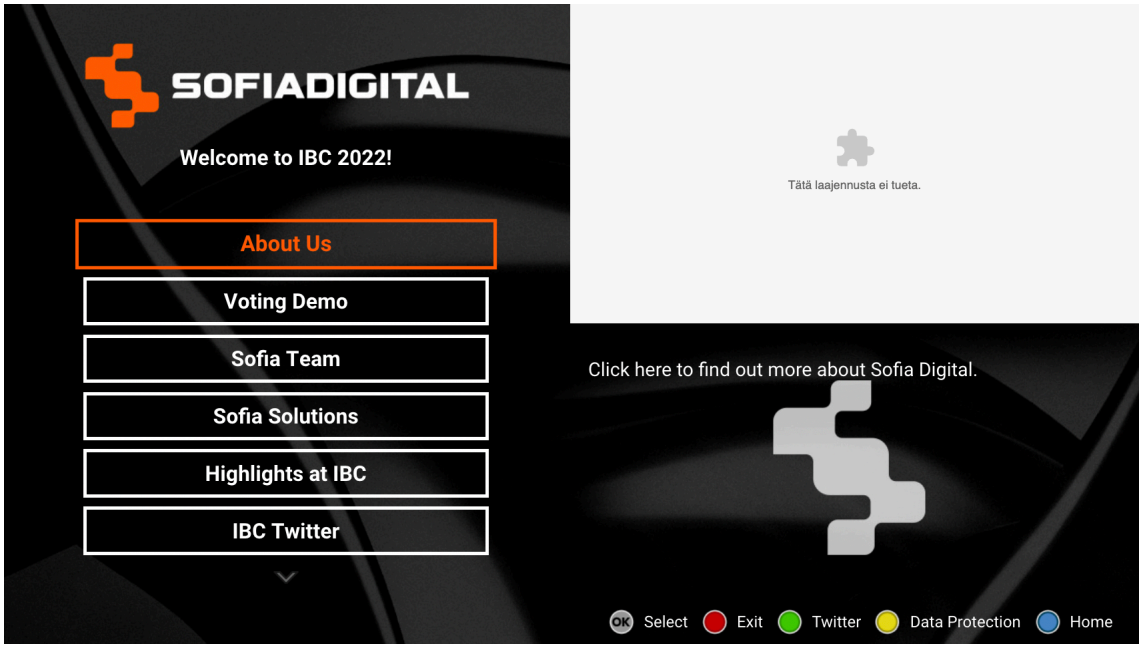
As this is a simple menu-based application with a limited number of UI elements, the focus management is handled simply by highlighting the border and/or background of the focused element, depending on the editor configuration, and navigation between focused elements is done using the directional keys. Other keys used in this application include the numeric keys and the colour buttons red, blue, green, and yellow.

In the browser preview mode, the remote-control keys are emulated by mapping them to similar keys on the keyboard: arrow keys for up, down, left, and right, enter for OK, backspace for the back key, R for the red button, B for blue, G for green, and Y for yellow.

The available colour buttons and their functions are shown in the application footer. These are automatically generated and cannot be customized by the user. The red button is always exit, either from the application or from a text overlay, such as the data protection dialogue. The green button opens the Twitter feed, and it is created and available whenever the application has a Twitter page in the configuration, whether hidden or visible. The yellow button opens the data protection dialogue, and it is only shown if either the data protection text or a list of cookies has been configured in the editor. The blue button is always available and returns the user to the front page of the application. The OK button will be visible whenever there is something on the page that the user can select.

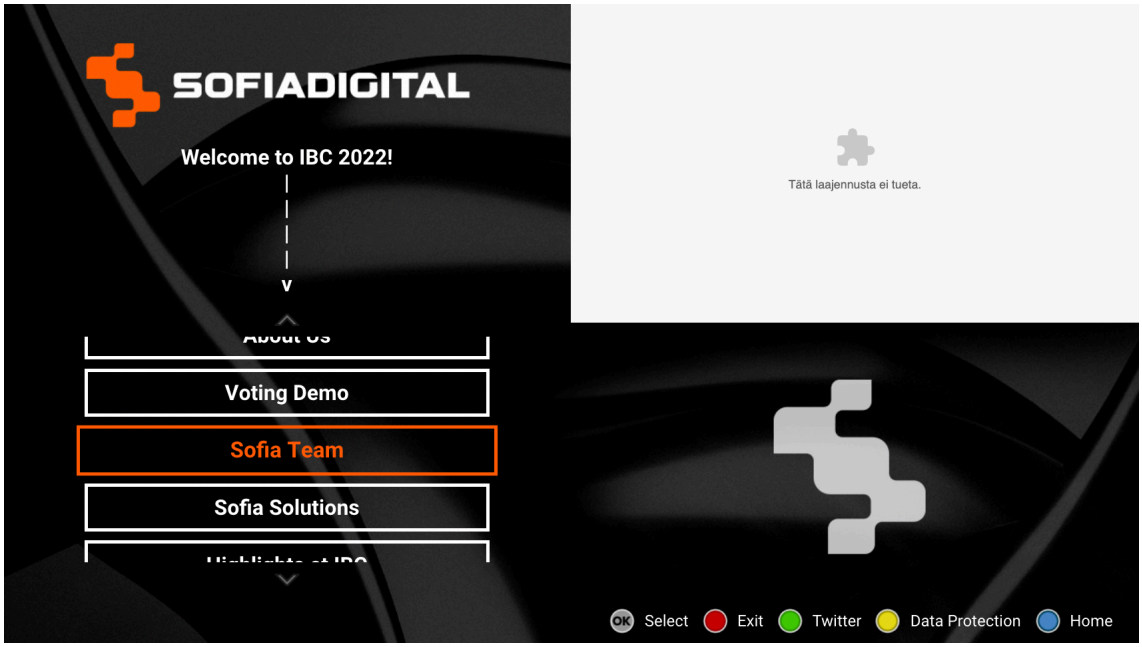
4.4.3 User interface

The user interface changes slightly depending on the page type, but generally it consists of three main sections: the left side containing the page description and menu, the top right quarter containing a scaled-down version of the currently playing television programme, and the bottom right quarter, which usually contains information about the currently focused menu item. An example of the front page or main menu view of the application can be seen in PICTURE 24.



PICTURE 24. Front page of the demo application.

The left-side menu height scales depending on the height of the page description, as demonstrated in PICTURE 25. If some menu items do not fit on the screen, small indicator arrows appear in the direction where the list can be scrolled up or down. A small image, called “app icon” in the editor, can also be inserted above of the page description, though it should be noted that in the example application the Sofia Digital logo is actually part of the application background image.



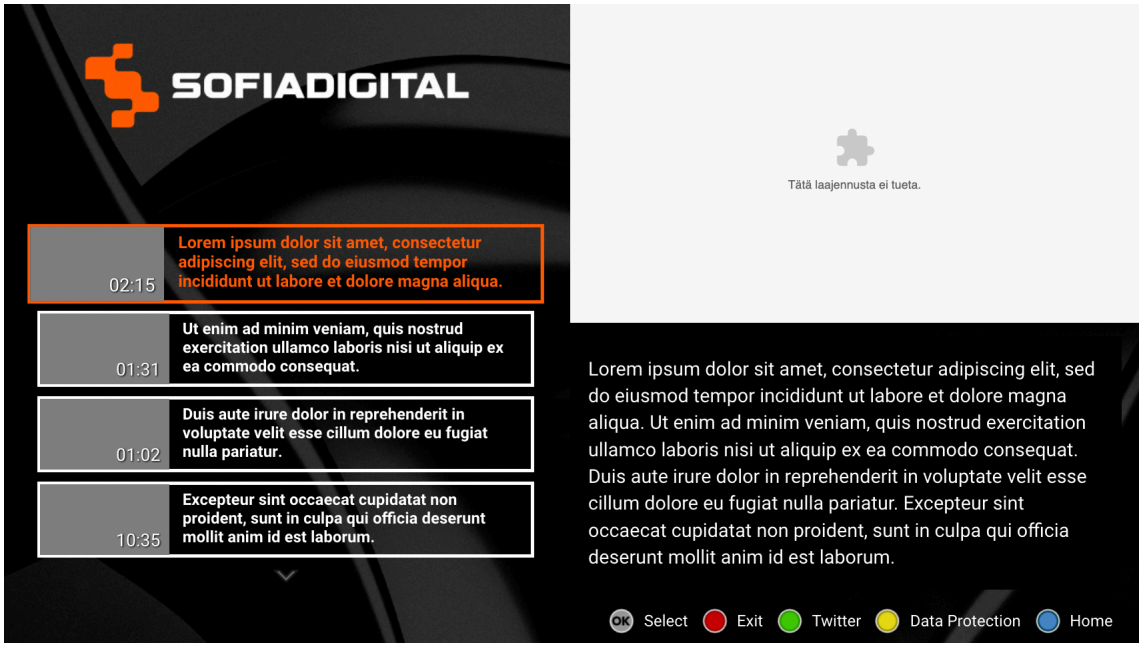
PICTURE 25. Menu scaling.

The video component in the top right quarter is a simple div containing a “video/broadcast” type object bound to the television feed using methods defined in the HbbTV Specification standard (HbbTV n.d.-c). It has no other functionality than passively playing the television programme on the side. Note that the video section will be blank in the browser preview version of the application, as no programme data is available in the browser.

In the menu layout, the bottom right quarter is intended to give further information about the focused menu item. This content is optional and can include text, an image, or both. The image will be aligned to the bottom right corner of the section and can either fit next to the text or extend under it. The text element has an automatic scrolling function which is triggered after a small initial delay when the full text does not fit on the screen.

The automatic scrolling functionality is implemented using a Vue mixin, which are essentially modules designed for the Vue component structure and allow injecting functionality into different components to avoid code duplication. Another example of mixin use in the application is sharing common functionality, such as store getters and focus change handlers, between the different route views. The automatic scrolling mixin can be attached to components that require automatic scrolling of any text element. It has methods that calculate the scroll position of the text element and scroll it down automatically until the bottom is reached, after which the scroll position is reset back to the top after a short delay, and the scroll cycle begins again.

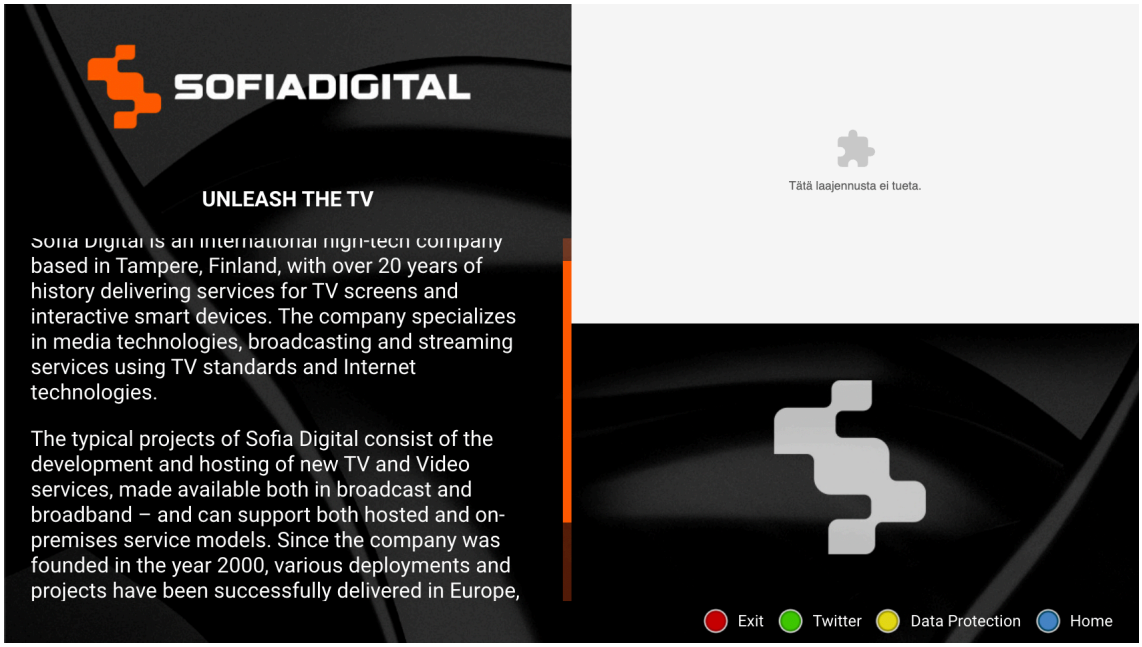
The same page layout is also used for the list, voting, and submenu page types. The clips menu (PICTURE 26) uses a slightly modified version of this menu structure with fewer customization options. It shows the preview image, duration, name, and description of each clip. The implementation of the clips functionality is not yet fully complete, but in the future, the user will be able to play content clips from this menu.



PICTURE 26. Clips menu.

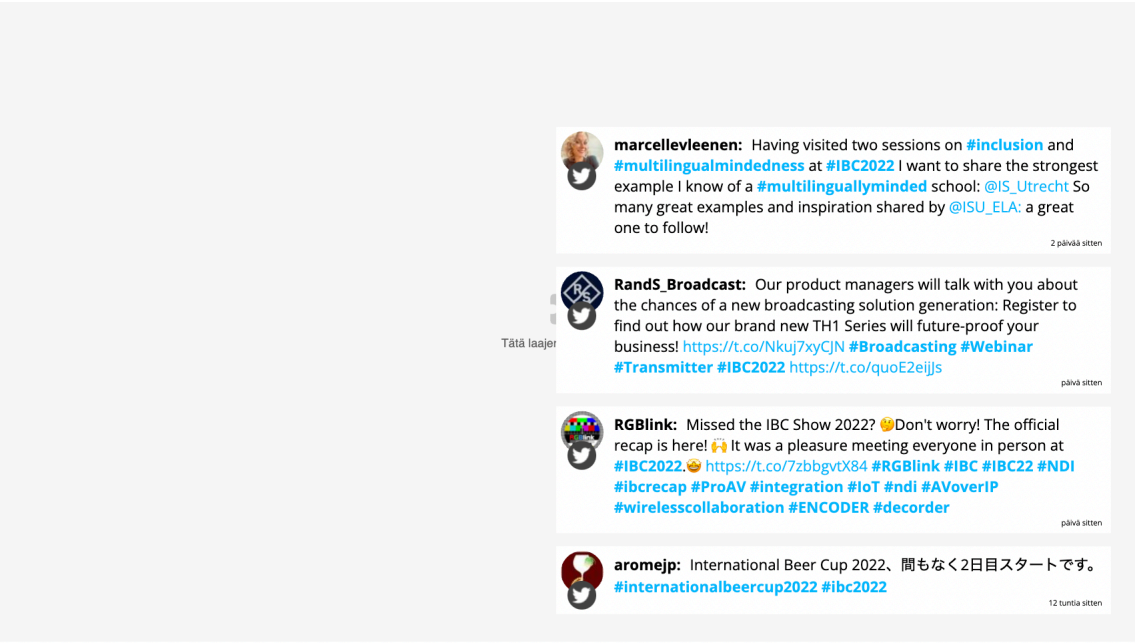
Text pages also use the same general layout, but instead of the menu structure, there is a scrolling text element on the left (PICTURE 27). This text element does not scroll automatically but is controlled by the user with the remote control. The

element on the bottom right quarter may either be empty or contain an image, as configured in the editor.



PICTURE 27. Text page.

Prize draws also have their own variation of the general layout, but they will be discussed in more detail in their own section. As mentioned, the Twitter feed does not utilize the regular application layout but is displayed directly on top of the unscaled television programme image, and it is customized through the poll application rather than the editor. An example of this view can be seen in PICTURE 28.



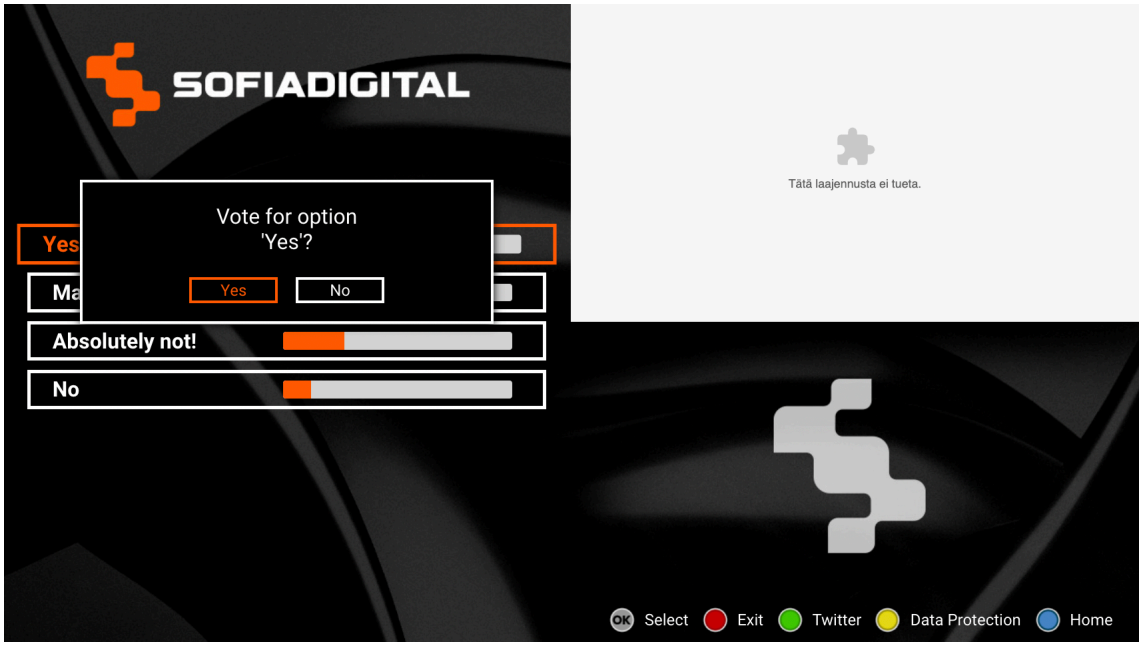
PICTURE 28. Twitter feed.

4.4.4 Languages

The application language can be selected in the editor. While most of the application text is set by the user, there are certain fixed elements, such as the colour button labels, dialogue messages and buttons, error toasts, and certain prize draw elements, which are automatically generated. The language of these text elements is controlled by the language selection. Currently, the only available language options are Finnish and English.

Languages are managed in the application using a custom-made localization module. This module has one function, which receives a “translation ID” string and returns the translation in the application language. The target language is read from the Vuex store, and the translations are read from separate dictionary files where each translation ID is paired with a translation. This system makes the addition of new language options fairly straightforward, as it mainly requires cloning one of the existing dictionary files, providing new translations, and setting it as a new dictionary option in the module. In case of missing translations, the English dictionary is used as a fallback.

The English and Finnish versions of the example application can be seen in PICTURE 29 and PICTURE 30, respectively. Both screenshots show the same voting page with the confirmation dialogue open. The voting options are user-defined text and thus remain unchanged, but the colour button labels and dialogue text change according to the language selection.



PICTURE 29. English localization example.

PICTURE 30. Finnish localization example.

4.4.5 Styles

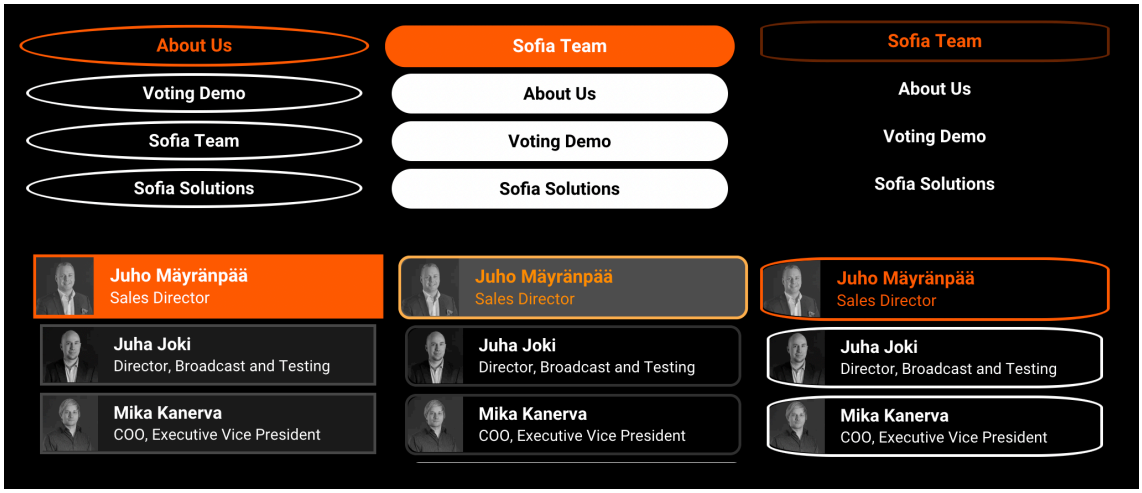
Styling may be the most difficult part of developing a HbbTV application. There are countless compatibility issues, and many CSS solutions will have unpredictable effects in older television models. The one saving grace is that the application window size is fixed and cannot be resized, so responsiveness is generally not a concern.

Regarding this project, another challenge was reliably reading all the application styles and page-specific style overrides set in the editor into the HbbTV application. The solution was to create a “dynamic styles” module, which has a function that can be called when the application configuration is loaded. It reads the styles from the configuration and builds a style sheet which is then injected into the application. It builds styles for all the customizable elements in the application and parses the hexadecimal colour values generated by the editor into the better supported RGBA format.

The page-specific style overrides are handled by creating a custom class for each page based on its position in the configuration and passing it to the style sheet builder function, which sets it as the parent class to all the CSS styles for that page. The same class name generation algorithm is then also used in the view mixin, from where the appropriate class name can be passed to each of the view components as the class for the top level div, which will bind it to the styles defined in the dynamically generated style sheet.

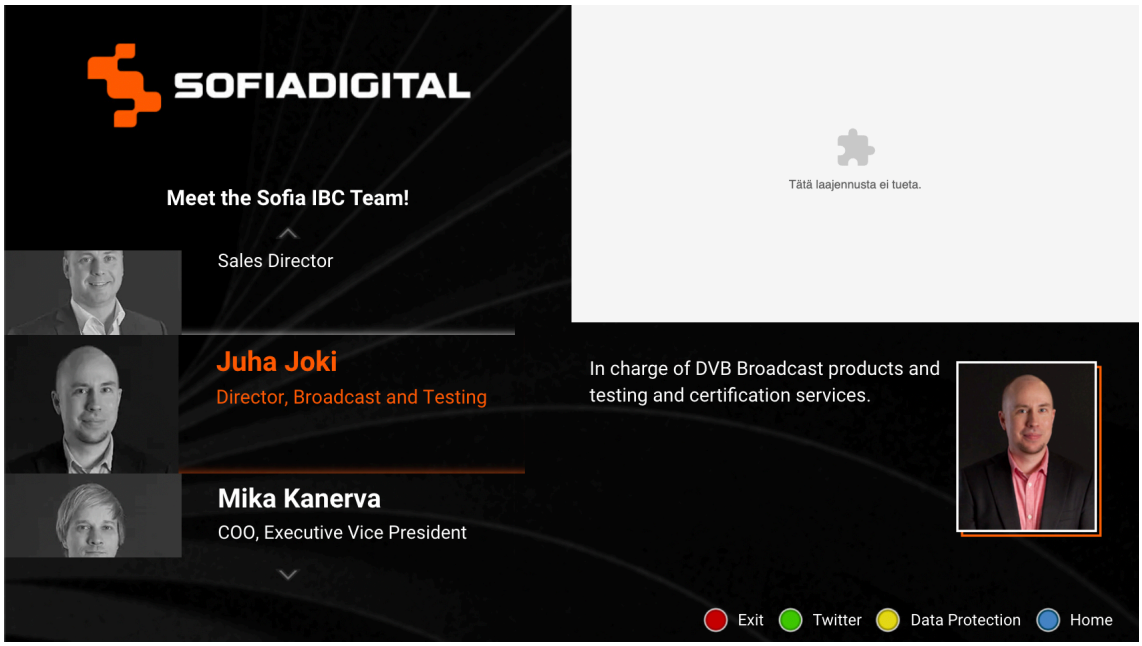
The same style builder module is also responsible for the application fonts. It attempts to read user-defined fonts from the configuration, then creates font-face definitions using that font, which are appended to the style sheet. Default fonts are used if no custom font is defined.

As discussed in the editor section, there are many different styling options available to the user. These include colours, images, and menu item shape and style. PICTURE 31 shows a few possible menu item variations for the example application theme.



PICTURE 31. Menu item style variations.

The menu item image style can also optionally be set as “elaborate”, which means that menu items which have an image attached to them will be shown as larger and with more elaborate styling. An example can be seen in PICTURE 32. If the menu item images were transparent, a small glow could also be seen behind the image. However, some compatibility issues were encountered when attempting to implement the glow, and further testing is still required to finalize this feature.



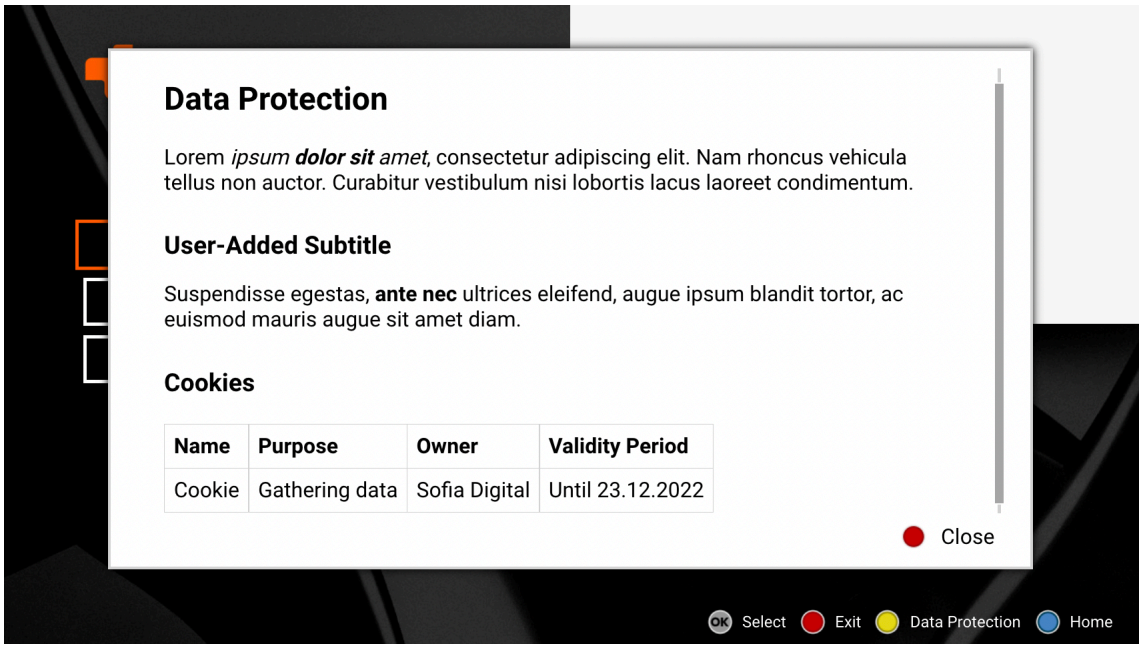
PICTURE 32. Elaborate menu items.

4.4.6 Formatted text

As explained in the editor section, certain text fields in the application use a special formatted text syntax, which allows adding headings, italics, and bolded text to the application text. These formatting options can also be nested so that, for example, a heading may have an italicized word in the middle, or a part of italicized text can have a bolded section for emphasis.

In order to add the required formatting tags dynamically, these text fields use the Vue “v-html” directive, which interprets text as HTML instead of plain text. This can be a dangerous directive when combined with user input, as it can leave the application vulnerable to script injection attacks. To prevent this, the formatted text is first sanitized to escape all HTML special characters, and only then the formatting syntax is converted into the desired styling tags.

One field which uses this special formatting is the data protection dialogue, which also conditionally adds a cookies section to the end of the user-defined text if cookie information has been added in the configuration. An example of the data protection dialogue showing some lightly formatted mock-up text and the automatically generated cookie table can be seen in PICTURE 33.

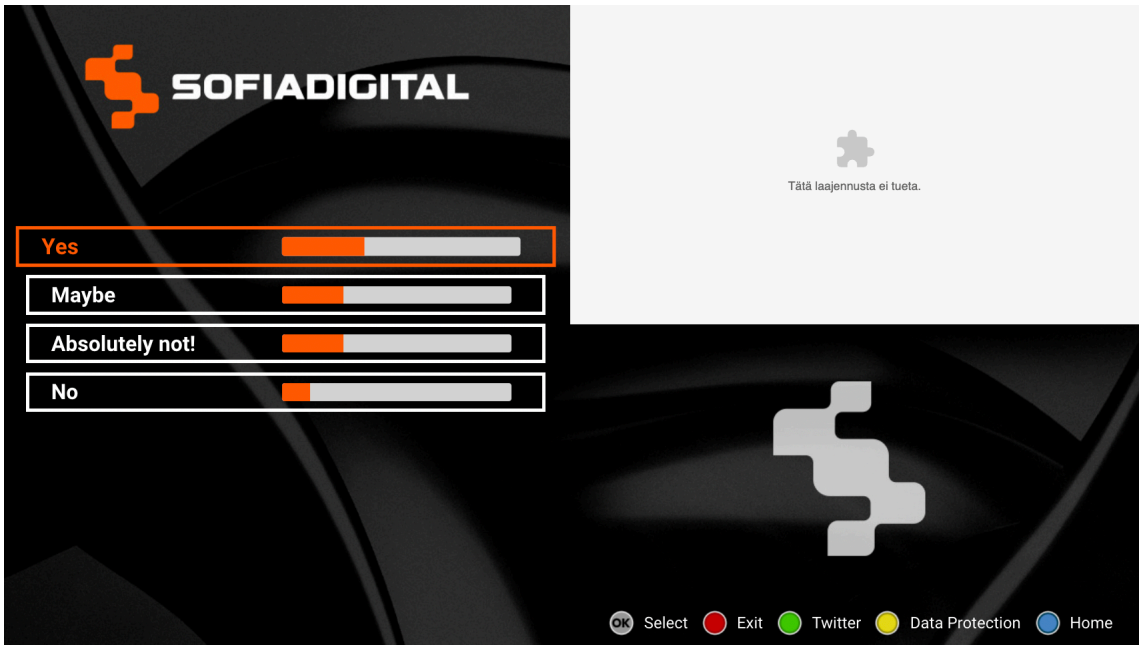


PICTURE 33. Data protection dialogue.

4.4.7 Voting

Basic voting pages are visually identical to lists, except the OK button appears in the footer to indicate that some kind of action on the list items is possible. The user can navigate between the options using the up and down buttons on the remote control and vote by pressing OK. A confirmation dialogue will be shown to help the user avoid accidental votes. Voting is not allowed in the browser preview mode.

If the “Show Results” option has been enabled for the voting page, the poll results will be visible to the user. This changes the menu item layout slightly to make room for the results bars, and a loading indicator may be shown while the results are being retrieved from the API. However, the basic user interface and voting functionality remain the same. PICTURE 34 shows one example of this type of voting page.



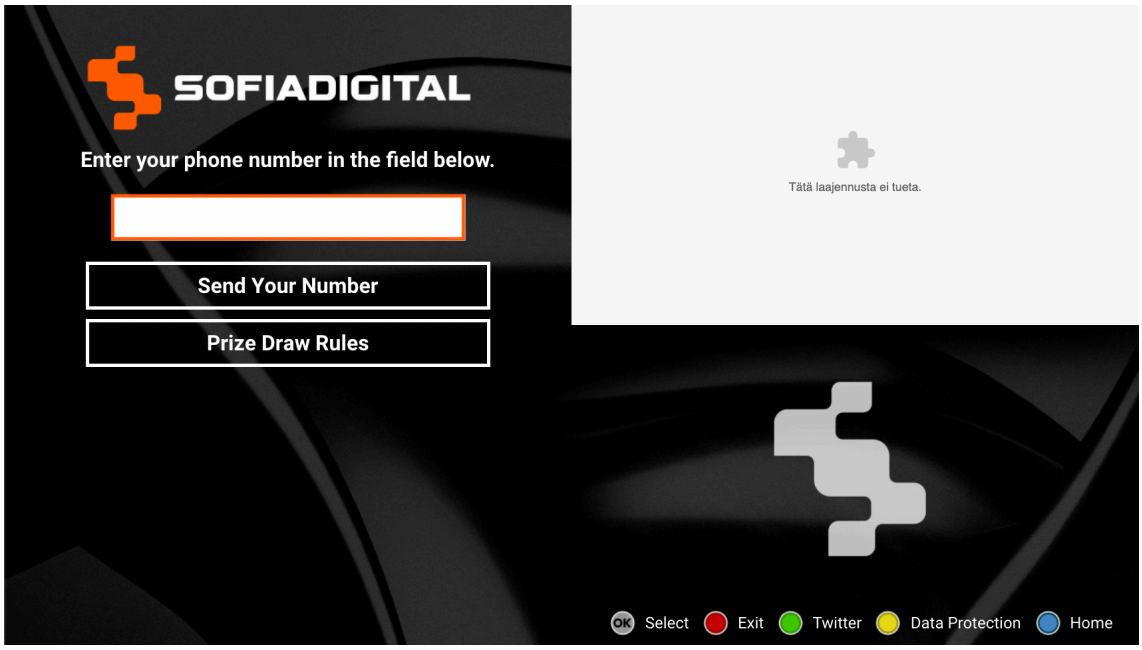
PICTURE 34. Voting page with the results visible.

In order to avoid revealing implementation information to bad actors, the exact details of the voting functionality are not documented in this thesis. However, the approximate process is that after the user accepts the confirmation dialogue to vote for an option, the HbbTV application performs certain checks to ensure that the poll exists and is configured to be available. Next, the vote information is sent

to the auxiliary backend where further checks are performed to ensure that the user is truly allowed to vote in the poll and has not exceeded the number of votes allowed in the poll. The auxiliary backend then sends back a response informing the HbbTV application of the result, and finally the user is shown a toast message informing them whether the voting was successful.

4.4.8 Prize Draws

Prize draw participation is also handled through the auxiliary backend, but the functionality is much simpler. Only participation via phone number is available at this time, so the draw page simply renders a text input which only accepts digits, and buttons for sending the number and reviewing the rules (PICTURE 35). The rules button is only rendered if rules for the draw have been configured in the editor. Additional text or image content can also be added to the right-hand side of the page, for example, voting instructions.

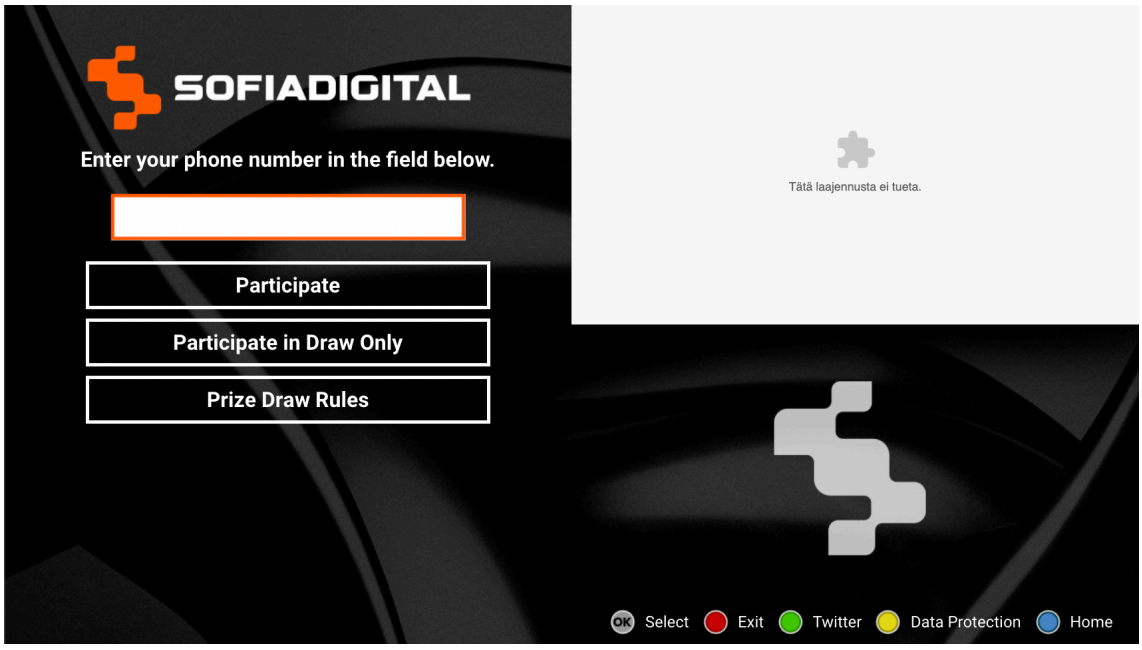


PICTURE 35. Prize draw page.

The phone number can be entered using the remote control's number keys while the phone number input is focused, and the back-button functions as backspace with its default navigation functions disabled. The application checks the length of the number and does not allow obviously incorrect entries to be sent. The user is also asked to confirm that they accept the prize draw rules on selecting "Send

Your Number”. The application then checks that the draw is configured to be currently active before sending the number. If not, an error message is displayed to the user.

If the marketing consent option has been selected in the configuration, two participation buttons are rendered instead of one (PICTURE 36). The one labelled “Participate in Draw Only” will take the place of the original button to send one’s number without any additional actions. The one labelled “Participate” will ask the user to also consent to marketing messages from the draw organizer. The marketing consent information is then sent to the auxiliary backend along with the number.



PICTURE 36. Prize draw page with the marketing consent option.

5 DISCUSSION

Overall, the project reached its goals. However, there is always room for improvement. For one, the use of pre-existing services caused some fragmentation of the project. If the project had been completely rebuilt from the ground up, the project structure could have been simplified significantly. For example, the the poll configuration and auxiliary backend functionality could have been bundled into the editor project. However, it did not realistically make sense to completely rewrite critical functionality when well-tested and stable implementations already existed.

Many features were also selected based on the needs of previous applications commissioned from the company, causing them to perhaps appear oddly specific for a generic application editor. For example, all the prize draw configurations were based on previous applications. Perhaps more universal options could have been implemented instead, but the ones chosen were prioritized due to previous and predicted future requirements.

The goal of the editor was to be accessible to non-developers, and this goal was more or less achieved. Internal user testing indicated that both developers and non-developers alike are able to use the editor to create and customize applications. However, many usability improvements could still be implemented. For example, the poll configuration is not entirely intuitive. Much of this is due to the involvement of the legacy poll application, but better user instructions could nonetheless be added.

The editor allows a high degree of customization. Because of this, some parts of the user interface remain somewhat cluttered, particularly on smaller screens. Better optimization for different screen sizes is definitely still required. Furthermore, many image-based user hints are still awaiting a redesign by a graphic designer to be made clearer and more aesthetically pleasing.

Usability could also be improved if the menu preview on the menu configuration page showed more information, such as the type of each menu item. The application landing page should probably also be visible (and perhaps configurable)

in some separate, singular location rather than having to search through each menu item to find what the landing page is set to.

The colour configurations could probably also benefit from a more “what you see is what you get” kind of approach, perhaps using some kind of simple updating preview to make the effect of each selector immediately obvious. Being able to override only certain styles, such as the background image, would also improve usability. This way, the user would not have to go through all of the style overrides to update them every time they make changes to the global theme.

The HbbTV preview page could also be made to update in real time without forcing the user to reopen it after each change. Similarly, poll application data could be refreshed automatically at certain intervals without the user having to refresh it manually. In fact, this functionality already exists in the results tab, so it would not be particularly difficult to implement. These data updates could also be separated from the poll results to give the user more control over updating and freezing the results data.

The login functionality is also still under development. The main improvements will be password protection and detaching the user login from the poll application. Currently, a poll application account is required to use the editor even if the application does not have any polls. Another potential future improvement is adding different user types for the same application with different access rights, for example, some users only being able to edit certain data and perhaps not being able to see the poll results.

Other pending improvements include updates to the configuration control. Currently, the editor does not allow the user to delete, rename, or save over older configurations. It is also not always clear to the user which configuration is currently loaded and whether changes have been made since the configuration was loaded. However, some of these updates would require significant structural changes in the application state management, so they have been placed lower on the priority list.

The feature selection could also still be expanded further. For example, more text and image alignment options could be added, as well as the possibility to generate simpler, single-page applications without the menu structure. Additionally, different user identification options could be added to the HbbTV application for things such as poll participation and even personalized content. However, account logins on television are often found cumbersome due to the remote-control navigation and may reduce engagement with the application, so any such functionality should be carefully designed to be as effortless to use as possible. The investigation of this is still in the early stages, but one possible solution might involve QR codes paired with a mobile application.

Similarly, the prize draw participation options could be expanded to allow other methods of participation beyond just the phone number, such as email. While typing an email address on a remote control would be much more difficult than a phone number, which can be inputted directly using the number keys, end users might be more willing to share an email address than a phone number, especially when marketing messages are involved.

Other potential features include more control over the functionality of the HbbTV application. For example, there could be an option where the poll results are only shown to the user after they have voted, or one where voting unlocks the prize draw page. Colour button functionality could also be allowed to be further customized, and certain actions could perhaps be attached to list items.

Meanwhile, pending updates for the HbbTV application include improving compatibility with older television models, implementing support for any new editor options, and adding better indicators for used and remaining votes on poll pages, especially where multiple votes per user are allowed. For the international markets, a more sophisticated localization system with better support for different languages and writing systems might also be required.

Furthermore, some work also remains to be done on the auxiliary backend and content delivery side of things, but these fall outside of the scope of this thesis project.

6 CONCLUSIONS

Due to the large scope of the project, the development of the application with the commissioning company will continue. However, the core functionality, which was the main goal of the thesis project, was completed, and though not yet released, the project has already been integrated into the commissioning company's Sofia Backstage® product family (Sofia Digital n.d.). A demonstration of the project was presented by the company representatives at the International Broadcasting Convention 2022 in Amsterdam, and it was also one of the finalists in the HbbTV Awards 2022 in the category "Best tool or product for HbbTV service development or delivery" (Clover 2022).

The editor is planned to initially be released in Finland and used for generating television show companion applications similar to the older applications it was based on. However, there is much room for expansion once the product is tested on the domestic market, as there is much less functionality tying it directly to a specific environment compared to the previous applications. It may also have potential for internal use for the company in the development of future HbbTV projects.

REFERENCES

Better Software Group. 2021. Differences in HbbTV and Smart TV – Pros and Cons. Published on 25.11.2021. Read on 16.10.2022. https://www.bsgroup.eu/blog/differences-in-hbbtv-and-smart-tv-pros-and-cons

Clover, J. 2022. Nominees announced for HbbTV Awards 2022. Published on 12.10.2022. Read on 15.10.2022. https://www.broadbandtvnews.com/2022/10/12/nominees-announced-for-hbbtv-awards-2022

Digita. N.d. HybridiTV-vastaanottimet. Read on 16.10.2022. https://www.digita.fi/antennitv/hybriditv/vastaanottimet

Fallahi, F., Joki, J., Teirikangas, J. 2019. HbbTV learning for beginners. Read on 10.9.2022. https://www.hbbtv.org/wp-content/uploads/2019/09/HbbTV-learning-for-beginners.pdf

Fischer, W. 2020. Digital Video and Audio Broadcasting Technology – A Practical Engineering Guide. Fourth edition. Cham: Springer Nature Switzerland AG.

Hasselström, J. 2018. Building an HbbTV application with ES7, React and Redux. https://medium.com/the-svt-tech-blog/building-an-hbbtv-application-with-es7-react-and-redux-612eaffc8aa

HbbTV. N.d.-a. 10 Year HbbTV Anniversary Special. Read on 17.9.2022. https://www.hbbtv.org/10-year-hbbtv-anniversary-special

HbbTV. N.d.-b. Deployments. Read on 17.9.2022. https://www.hbbtv.org/deployments

HbbTV. N.d.-c. Specifications. Read on 17.9.2022. https://www.hbbtv.org/resource-library/specifications

HbbTV. N.d.-d. Developer Portal. Read on 16.10.2022. https://developer.hbbtv.org/

Joki, J. 2022. HbbTV: enemmän kuin telkkari. Published on 26.04.2022. Read on 16.10.2022. https://ficom.fi/ajankohtaista/uutiset/hbbtv-enemman-kuin-telkkari

Langendijk, M. 2021. Smart-TV Development: The Basics. Published on 27.10.2021. Read on 16.10.2022. https://mlangendijk.medium.com/smart-tv-development-the-basics-5ec22a9ea2ad

Malhotra, R. Hybrid Broadcast Broadband TV: The Way Forward for Connected TVs. IEEE Consumer Electronics Magazine, vol. 2, no. 3., 10-16.

Mautilus. 2018. HbbTV Development – How to Make an App Compatible on Every Device. Published on 6.2.2018. Read on 16.10.2022. https://www.mautilus.com/blog/hbbtv-apps-development-how-to-make-an-app-compatible-on-every-device

Pratt, M. 2022. What is a smart TV? Best smart TVs for 2022. Updated on 12.9.2022. Read on 16.10.2022. https://www.which.co.uk/reviews/televisions/article/what-is-smart-tv-aNeqk6F0RAAn

Sofia Digital. N.d. Programme-specific HbbTV applications. Read on 15.10.2022. https://sofiadigital.com/our-offering/hbbtv/programme-specific-hbbtv-applications

TM Broadcast. N.d. HbbTV: A decade driving the digital TV transformation. Published on 30.12.2020. Read on 16.10.2022. https://tmbroadcast.com/index.php/hbbtv-decade-driving-digital-tv-transformation

Vue CLI. 2022. Browser Compatibility. Updated on 4.9.2022. Read on 16.10.2022. https://cli.vuejs.org/guide/browser-compatibility.html