



TPF
Technologieplattform

Fabrikstrasse 8
Büro A209
3012 Bern

031 6315513

SelfHelp WebApp - Create Web Applications Serving as a Platform for Research Experiments

Tutorial for version v1.0.0

Simon Maurer - simon.maurer@humdek.unibe.ch
1st February 2019

Contents

1	Introduction	3
2	Creating a New Page	4
2.1	Add Content to the Page	5
2.2	Create a Menu of Pages	6
2.3	Create a Navigation Page	8
2.3.1	Linking to Navigation Sections	9
3	Symbols with Special Meaning	12
4	User Input	13
4.1	Store User Data to the Database	13
4.2	Fetch User Data from the Database	14
4.3	Conditional Content	15
5	User Management	16
5.1	Users	16
5.2	Groups	16
5.3	Registration of Users	16
6	Extending the WebApp	17

Chapter 1

Introduction

This documentation refers to the TPF project [SLP-selfhelp](#). The project aims at providing a tool that allows to create a web application that serves as a platform for research experiments. The basic concept is as follows:

Pages are organised as a collection of sections that are rendered on the page, one below the other. Sections have different styles which define the appearance of the sections. Depending on the style of a section, the section has different fields which define the content of the section. The value of a field can be a simple plaintext or a collection of child sections which have their own styles and children.

Currently available styles include, but are not limited to, alert boxes, buttons, card containers, forms, media elements, tabs, lists, and support for markdown texts.

The app is designed in such a way that it can be extended with new styles or custom components without having to modify existing code.

The main purpose of this document is to provide a step-by-step tutorial of how to work with the admin section of the SelfHelp WebApp.

For an overview of existing styles and their fields refer to [this demo page](#)¹. Instructions for configuring the web server can be found on [this page](#)². For the documentation of the source code refer to [this doxygen page](#)³. Finally, [this page](#)⁴ provides some information about possible extension and customization options of the WebApp.

In the following it is assumed that a server is up and running and an instance of this WebApp is made available. After a clean setup an account `admin` is activated which can be accessed with the password `admin`;

¹ <https://selfhelp.psy.unibe.ch/demo/styles>

² <https://selfhelp.psy.unibe.ch/demo/configs>

³ <https://selfhelp.psy.unibe.ch/demo/doc/doxygen/html/index.html>

⁴ <https://selfhelp.psy.unibe.ch/demo/extend>

Chapter 2

Creating a New Page

In order to create a new page the `select` and `insert` access right of **Page Management** are required. By default users in the group `admin` and `experimenter` are granted the mentioned access rights.

Once logged in, navigate to the *Content Management System (CMS)* in the menu *Admin* and click the button *Create New Page* in the top left corner.

Create New Page

Page Properties

Keyword

demo

The page keyword must be unique, otherwise the page creation will fail.

Page Type

☒ Sections ☐ Navigation ☐ Component ☐ Custom

The page type specified how the page content will be assembled. It is recommended to either use the type `Sections` or `Navigation`. Pages of type `Component` and `Custom` require PHP programming and cannot be handled by the CMS.

☒ **Header Position**

0 contact

1 New Page

When activated, once the page title field is set, the page will appear in the header at the specified position (drag and drop). If not activated, the page will **not** appear in the header.

Url Pattern

/demo

This is set automatically. If you know what you are doing you may overwrite the value. Refer to the documentation of [Altorouter](#) for more information.

☐ **Advanced**

Create **Cancel**

Figure 2.1: The *Create New Page* form.

This opens the *Create New Page* form (see Figure 2.1) where properties of the new page can be entered:

Keyword This must be a unique name which is used to identify a page internally (e.g. `demo`). This is also the name that will be used in the CMS when a link to this page must be shown. Note that this field is not visible for a user without

access to the CMS (e.g. a subject).

Page Type The page type specified how the page content will be assembled. Choose type **Sections**.

Header Position Enable this field to make the page appear as a menu-point in the navigation bar of the WebApp. Drag and drop the item **New Page** to the location you want it to appear. Note that the page will only appear as a menu item once the `title` field is set (this must be done after the page creation process).

Headless Page (advanced only) This allows to select whether the page will be rendered with (default) or without header and footer.

Protocol (advanced only) The protocol specifies how a page is accessed. Do not change this.

Url Pattern (advanced only) This field describes the url path of how the page will be accessible. By default the keyword is used.

Advanced Set this flag if you wish to specify advanced options. This includes

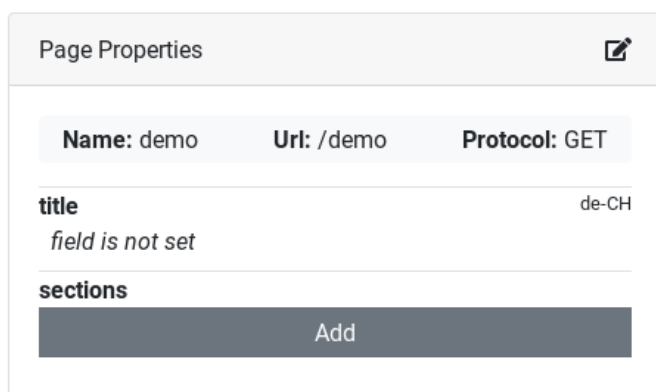
- the selection of the *Page Type* **Component** or **Custom**.
- the selection of the *Protocol*.
- changing the *Url Pattern*.
- changing the *Headless Page* parameter.

Click the button **Create** to create the page and on success the button **To the new Page** to return back to the CMS with the new page selected.

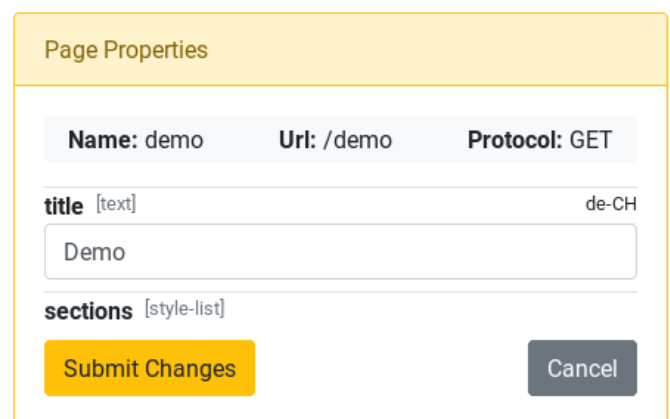
2.1 Add Content to the Page

In order to add content to a page the update access right of **Page Management** is required. By default users in the group `admin` and `experimenter` are granted the mentioned access rights.

In a first step let's define the title of the page `demo`. The title is used as label for the menu item in the navigation bar of the WebApp and will appear in the browser tab. To set the page title select the pen symbol in the top right corner of the *Page Properties* card. Doing so will change the card colour to yellow and provide an input field where the title can be entered. If you just created a new page you will already be in edit mode (which is indicated by the yellow border of the *Page Properties* card). Change the field `title` (e.g. to `Demo`) and click **Submit Changes** to make the change permanent. If during the creation of the page the option *Header Position* was enabled, the title `Demo` will now show up in the header at the specified position.



(a) View mode.



(b) Edit mode.

Figure 2.2: The *Page Properties* card.

Adding content to the page is done by creating and adding sections to the page. This can be done by clicking on the button **Add** below the field `sections` in the *Page Properties* card. This opens the *Add Section* form where the name and the style of the new section can be defined. The name serves only to let you identify the section at a later point and can be any

string. It is, however, recommended to use a logic structure in naming sections (it is a good idea to include the page name into the section name). Note that the style is always appended to the section name. The style defines how the section will look like and what type of content can be assigned to it. Let's choose the style `container` which is used as wrapper for other styles to be added to the page. This helps to manage the spacing at the edges of the page.

The *Add Section* form, as depicted in Figure 2.3, has several cards: The *Add Section* card is all that is needed to create a new section. However, given that there are quite a few styles to choose from the card *Style Selection Helper* may come in handy. This card aims to group styles into different categories and provides a very short description of each available style. The two cards *Select an Unassigned Section* and *Select an Existing Section* allow to choose already existing sections without creating a new one where the former lists sections that are assigned to no parent and the latter lists all user-created sections.

Figure 2.3: The *Add Section* form.

Click **Add Section** to create the new section which will return you to the CMS with the new section selected in edit mode (the card *Section Properties* is yellow).

The next step is to edit the properties of the section. A container section only has the two fields `CSS` and `is_fluid`. Enable the field `is_fluid` to make the container span over the whole page. The field `CSS` is special in the sense that every style has this field. It allows to specify `CSS` classes (separated by spaces) that will be added to the root `html` tag of the style. Add the bootstrap class `my-3` to the field to add some spacing at the top and the bottom of the container (refer to the [Bootstrap documentation](#) from more details).

Now continue to add sections to the container you just created. For example:

- A section of style `jumbotron` which, itself, holds a section of style `markdown`. In the field `text_md` of the `markdown` section use `markdown` syntax to add a title and a paragraph of text.
- A section of style `alert` which has a child section of style `heading` and another child section of style `plaintext`.
- A section of style `card` which has a child section of style `quiz`.

Doing all of this and tweaking the available fields of the different styles should net you with something resembling Figure 2.4

2.2 Create a Menu of Pages

Up to this point we created a section page and added the link to this page to the navigation bar. It is also possible to create a menu item with multiple page links. It is, however, not possible to change the position or hierarchy of a page once it is

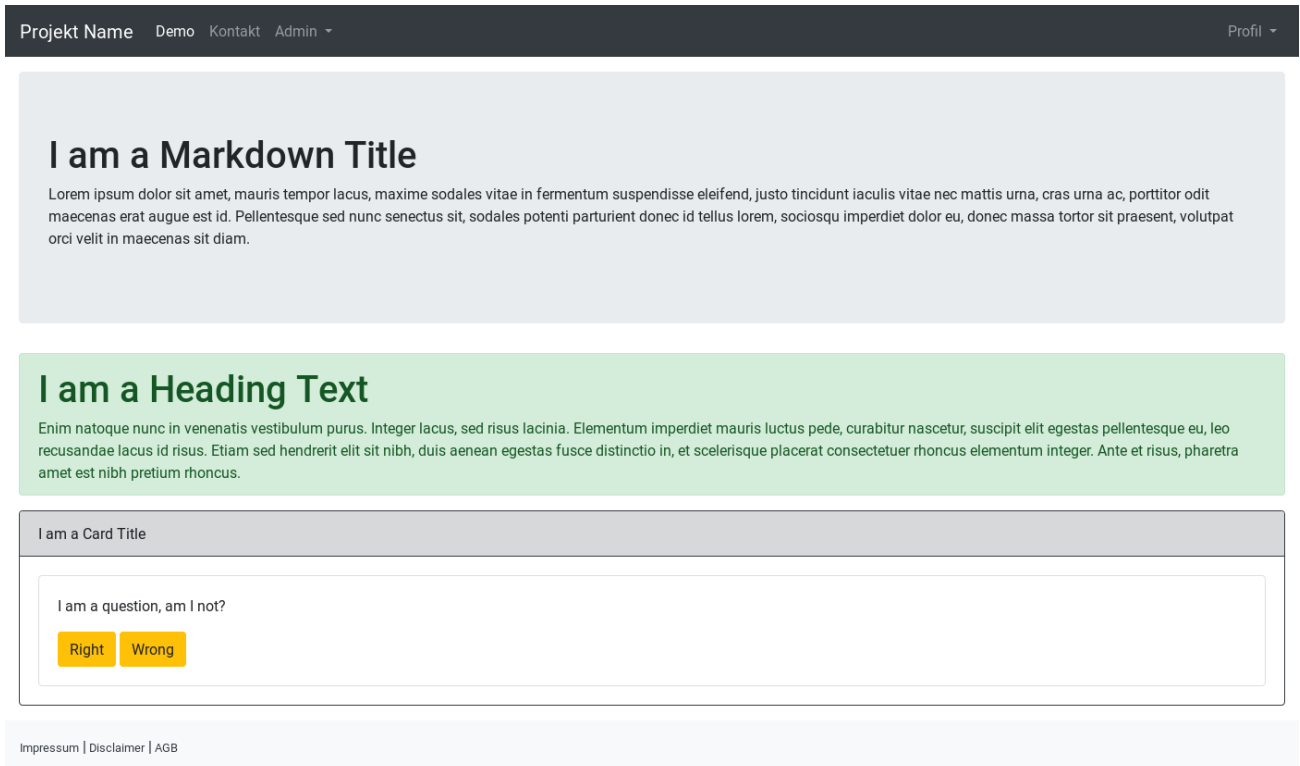


Figure 2.4: A demo page of type **Sections**.

created. A workaround is to remove the page and create a new one that fits those requirements. Note that sections remain untouched when a page is deleted, hence, recreating a page with its initial content is not a big deal.

In order to delete a page the `delete` access right of **Page Management** is required. By default users in the group `admin` and `experimenter` are granted the mentioned access rights.

As a demonstration let's create a dropdown menu in the navigation bar and add the content we created before as a child element:

First, the recently created page `demo` has to be removed. To do this select the page in the *Page Index* of the CMS, open the card *Delete Page* (red), and click the button **Deplete Page**. This opens the *Delete Page* form where the keyword of the page has to be entered into the input field to confirm the deleting process. Click **Delete Page** and on success click **Back to CMS** to return to the CMS.

Now, create a new page which will only serve as the root menu item in the navigation bar. For this purpose I propose to append the string `-link` to the keyword of the new page (e.g. `demo-link`). Enable the field `Header Position` and choose the appropriate position. Leave the remaining fields untouched and click **Create**.

Back in the CMS define the title of the page in the *Page Properties* card as described in the beginning of Section 2.1. Once this is done click the button **Create New Child Page** in the card *Create New Child Page*. Once again the *Create New Page* form is opened where we can recreate the page `demo` as described in Section 2 in the beginning of this chapter. Note that when enabling `Header Position` only the new page is shown. This is because this page is the first child page in this menu and the position refers to the current hierarchy level of the page (i.e. to the children of the page `demo-link`). Back in the CMS define the title of the page and click the button **Add** below the field `sections` to add the sections we created before.

Once the *Add Section* form opens you will notice that on the right in the card *Unassigned Sections* the container section we create before is listed. Select this section and click **Add Section** which will restore the page as it is represented in Figure 2.4 with the exception that now, the page is reachable through a dropdown menu in the navigation bar.

2.3 Create a Navigation Page

Often, it might be interesting to navigate from page to page by clicking on buttons **Next** and **Back** or by selecting the corresponding page in a list. To achieve this the page type `Navigation` can be used. A navigation page is a special type of section page where only one immediate child section (which can hold any number of child sections) is rendered at a time.

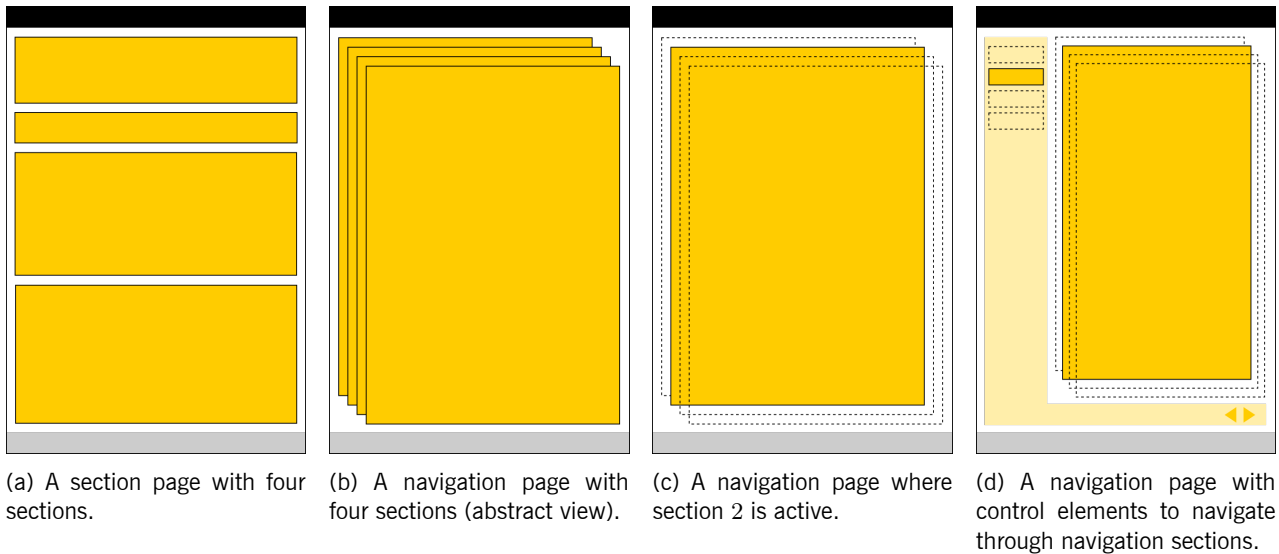


Figure 2.5: A conceptual representation of a section page and a navigation page.

Figure 2.5 aims at visualizing this concept. Figure 2.5a shows a simple section page with four immediate child sections. Figure 2.5b shows a representation of a navigation page with four immediate child sections. The sections of the section page are represented one below the other. However, the sections of the navigation page are represented like they were stacked on top of other like a card deck. In the following, such sections of a navigation page are called navigation sections. Note that only one navigation section of a navigation page can be visible. This is indicated on Figure 2.5c. A predefined set of buttons can be rendered on a navigation page which allows to navigate through all navigation sections associated to this page. Also, a menu can be rendered on a navigation page or any other page which allows to navigate through navigation sections and jump to a specific section. This is indicated on Figure 2.5d.

A navigation page has several characteristics:

- A navigation page can only be rendered if the navigation section to be displayed is specified (this is done via the url).
- Because of the previous point a navigation page always points to its first entry when displayed as a menu in the header.
- A navigation section can only have the style `navigationContainer`. However, children of such a section can have any style.

To demonstrate how a navigation page works, let us create such a page.

We start by creating a new page of type `Navigation`. Let this page be a child of the page `demo-link`. There is no real importance to this but it helps to keep things grouped together in the CMS. So, select the page `demo-link` and click the button **Create New Child Page**. Set a keyword (e.g. `demo-nav`) and set the type to `Navigation`. Note that the pattern `[i:nav]` is appended to the `Url Pattern`. This pattern will be replaced by the id of the selected navigation section to be displayed on the navigation page.

Back in the CMS the page fields can be edited. As with section pages, the field `title` defines what will be displayed in the browser tab. The field `css_nav` allows to add css classes to the navigation list that is displayed to the left of a navigation page. The fields `label_back` and `label_next` will be the labels of the navigation buttons **Next** and **Back**, respectively. These buttons will only be displayed if the field `has_navigation_buttons` is enabled. The navigation menu can be enabled/disabled with the field `has_navigation_menu`. The field `is_fluid` defines whether the content is stretched to the whole of the page or not. For more information about the rest of the available fields refer to the [nestedList style documentation](#).

In order to add a stack of navigation sections to this navigation page the **Add** button below the field `navigation` can be used.

Note that a navigation page has a field `sections` (which we already know from section pages). This allows to add sections that are rendered before the selected navigation section and will remain the same, independent of the navigation section that is currently displayed (refer to Figure 2.8 for an example).

Let's add some navigation sections: Click the **Add** button below the field `navigation` which will open the *Add Navigation Section* form. This is a simplified *Add Section* form where only a name can be chosen. Note that the style is fixed to `navigationContainer` and cannot be changed. Set the name (e.g. `demo-nav1`) and click **Add Section** to complete the process. In the Section Properties card, the field `title` serves as title for the navigation list item representing the navigation section wrapped by this container. The field `text_md` is rendered at the top of the navigation section. Note that the special string `@title` can be used to render the field `title` within the field `text_md`. Finally, the field `CSS` allows to add CSS classes to the navigation container section to modify its visual aspects.

As an example set the field `title` to `Nav 1` and the field `text_md` to `# Demo @title` (the `#` symbol is markdown syntax for a heading of level 1) and click **Submit Changes**.

In order to truly see how navigation pages work we have to add some more navigation sections. Do this by selecting the page `demo-nav` and adding two more navigation sections with the titles `Nav 2` and `Nav 3`.

2.3.1 Linking to Navigation Sections

Up to this point we created a navigation page with three navigation sections but have currently no way of accessing them. As noted before, in order to display a navigation page an id of a navigation section needs to be specified in the url. Let's assume that the keyword of the navigation page is `demo-nav`, the base path of the WebApp is `https://selfhelp.psy.unibe.ch/myapp`, and the id of the navigation section we want to display is `n` (Note that the id of a section is the last number in the url when selecting this section in the CMS). The url to access this navigation section would then be `https://selfhelp.psy.unibe.ch/myapp/demo-nav/n`.

Obviously, it is rather cumbersome to reach a navigation page this way which is why there is a mechanism to automatically generate these links. There exist three possibilities to generate links to a navigation section:

1. When checking the *Header Position* during the page creation step of a navigation page a link is generated which points to the first navigation section of the navigation page.
2. When using a link style (e.g. styles `link` or `button`), the field `url` can be set to a pattern of the specific form `#<nav_page>/<nav_section>` where `<nav_page>` is the keyword of the navigation page and `<nav_section>` the name of the navigation section. E.g. assuming that the navigation page keyword is `demo-nav` and the navigation section has the name `demo-nav1-navigationContainer`, to access the navigation section the field `url` would be set to `#demo-nav/demo-nav1-navigationContainer`.
3. A list style `nestedList` allows to be linked to a navigation page which will automatically generate the list of all navigation sections attributed to the navigation page. This is accomplished by setting the field `items` to the pattern `{"nav_page": "<nav_page>"}` where `<nav_page>` is the keyword of the navigation page to be linked. E.g. assuming that the navigation page keyword is `demo-nav`, to generate the list of links to all navigation sections the field `items` would be set to `{"nav_page": "demo-nav"}`.

To demonstrate how to link the navigation page `demo-nav` and all its navigation sections, let's use a nested list: We can achieve this by adding a new section to the page `demo` we created in the beginning. Select the page `demo` and add a new child section of style `nestedList` to the page:

1. Select any container section or the page itself and click **Add** below the field `children` or `sections`, respectively.
2. Select the style `nestedList` (from style group `List`).
3. Enter a name (e.g. `demo-nav-link`).
4. Click **Add Section**.
5. In the card *Section Properties* fill out the field `items` with the content `{"nav_page": "demo-nav"}`. This is a special `json` string which tells the nested list to go and fetch all navigation sections of the navigation page `demo-nav`.

6. Leave everything else as is and click **Submit Changes**.

As a result you should get something similar to what is represented in Figure 2.6.

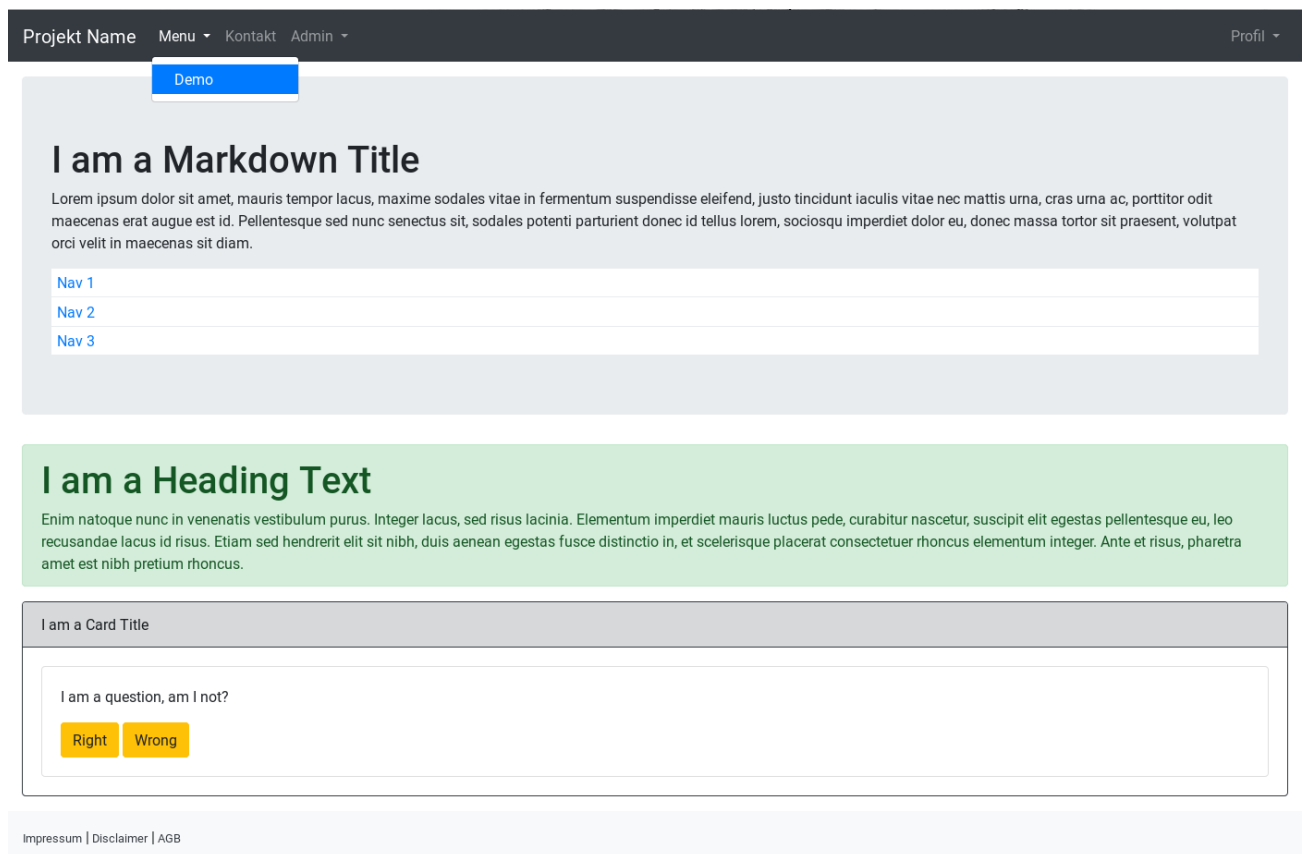


Figure 2.6: A demo page of type **Sections** with a list of navigation items added to the jumbatron.

The navigation page `demo-nav` is reached by clicking on any of the items displayed in the nested list. The result should look similar to Figure 2.7.

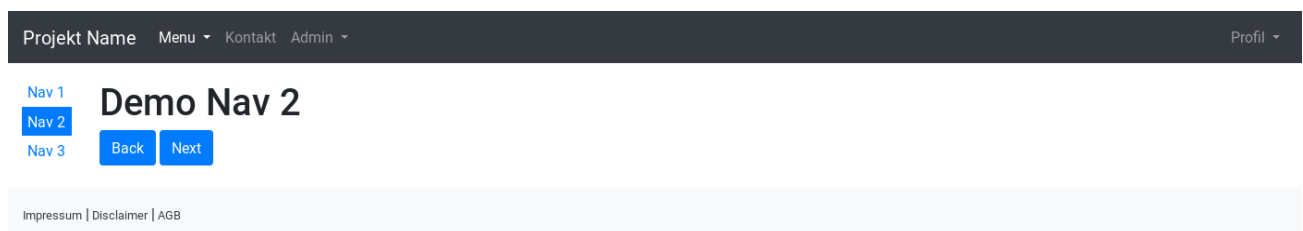


Figure 2.7: A demo page of type **Navigation** with a list of navigation items displayed on the left and navigation buttons displayed at the bottom of the page.

Note that sections that are added to the navigation page `demo-nav` through the filed `sections` will appear on top of each navigation section as demonstrated in Figure 2.8.

Also note that navigation sections can be arranged hierarchically by using the field `navigation` in the card *Section Properties* of a navigation section and adding a child navigation section.

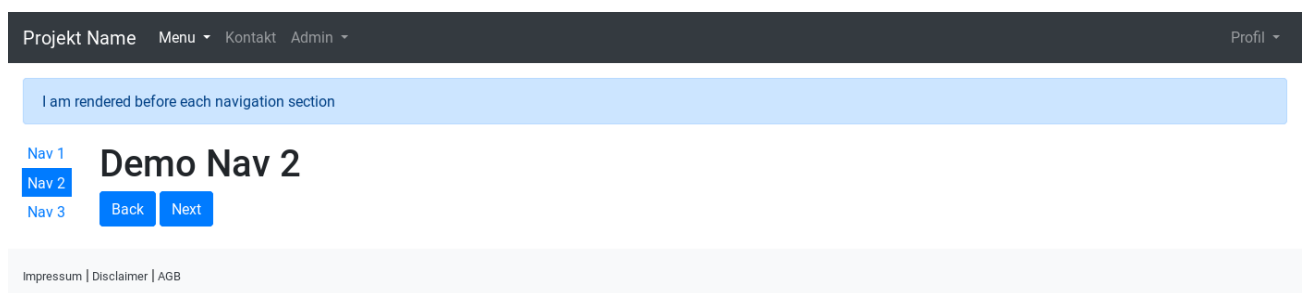


Figure 2.8: A demo page of type **Navigation** where before each navigation section the same section is displayed (the blue alert box).

Chapter 3

Symbols with Special Meaning

In order to facilitate the usage of certain styles several keywords can be used to render user-specific data, easily link to pages, or refer to stored files. The following list provides an overview of all available keywords in the different styles:

- In any field of any style the following keywords can be used:
 - `@user` This keyword is replaced with the name the user provided during the validation process.
 - `@project` This keyword is replaced with the title of the page `home`.
- In the styles `form`, `link` and `button` links can be generated if a special syntax is used:
 - `#self` refers to the current page.
 - `#back` refers to the page that was visited previously. Note that if the previous page cannot be found `#back` is equivalent to `#home`.
 - `%<asset_name>` links to a asset where `<asset_name>` is the exact name of the asset.
 - `#<page_name>` links to a page where `<page_name>` is the keyword of the page.
 - `#<page_name>#<wrapper_name>` links to an anchor on a page where `<page_name>` is the keyword of the page and `<wrapper_name>` is the name of a section that is used as an anchor. Note that the anchor section must be of one of the following styles for this to work: `alert`, `container`, `card`, `form`.
 - `#<nav_page_name>/<section_name>` links to a navigation section on a navigation page where `<nav_page_name>` is the keyword of the navigation page and `<section_name>` is the name of the navigation section.
 - `#<page_name>/<section_name>#<wrapper_name>` links to an anchor on a navigation section on a navigation page where `<nav_page_name>` is the keyword of the navigation page, `<section_name>` is the name of a navigation section, and `<wrapper_name>` is the name of a section that is used as anchor. Note that the anchor section must be of one of the following styles for this to work: `alert`, `container`, `card`, `form`.
- The field `text_md` of the style `navigationContainer` allows the keyword `@title` to refer to the field `title` of the navigation container.
- The field `condition` of style `conditionalContainer` allows the keyword `@<form_name>#<field_name>` to refer to the user data of a specific input field `<field_name>` of a specific form `<form_name>`.

Chapter 4

User Input

The SelfHelp WebApp provides several styles to work with user input. The style `formUserInput` can be used to store data, entered by a user, to the database. The style `showUserInput` is a simple style that allows to display data that was entered by a user via a specific section of style `formUserInput`. Finally, the style `conditionalContainer` allows to evaluate user input and display content depending on the outcome of the evaluation.

4.1 Store User Data to the Database

The Selfhelp WebApp allows to create html forms where user input is automatically stored to the database. In order for a page to be allowed to send data to the server the following two conditions need to be met:

1. The POST protocol needs to be enabled when creating a new page. This is enabled by default. The protocol of a page is set when creating a new page (advanced options).
2. The page requires input fields, wrapped by a form such that users can enter and submit their data. Forms and their input fields are added to a page by adding sections with styles from the groups `Form` and `Input`, respectively.

There are two ways of how user data is stored to the database automatically:

1. Data is continuously updated. In this case an input field always shows the current, user-specific content and whenever this content is changed by the user, the old values are overwritten in the database. This way of storing data is intended for entries where data is constantly edited and revised.
2. Data is stored incrementally. In this case each time a set of data is submitted to the server, a timestamp is attached to the data. For each submission a new entry is created in the database which allows to store multiple entries, made at different points in time. This way of storing the data is intended for log-like forms where a user is asked to enter a specific set of data every day.

As a demonstration let's add a new section page (set *Page Type* to *Sections*) with keyword `user_input` and title `User Input` to the demo application. Remember to check the *User Input* checkbox (in order to allow user input) and the *Header Position* checkbox (to place the page in the header) in the *Create New Page* form.

Next, add sections to this page to get a similar result as represented in Figure 4.1. This is accomplished by adding the following sections hierarchically to the page `user_input`:

- a container with `css = my-3` and `is_fluid` enabled
 - a card with `title = Diary`
 - * a `formUserInput` with `name = diary`, `alert_success = Well done`, and `label = Submit`
 - a `textarea` with `name = diary-thought` and `label = My Thoughts`
 - a `textarea` with `name = diary-action` and `label = My Actions`

- a card with title `Log` and `css = mt-3`
 - * a `formUserInput` with `name = log`, `alert_success = Well done`, `label = Submit`, and with `is_log` checked
 - an input with `type_input = time`, `name = log-alarm`, and `label = Set my Alarm to`
 - an input with `type_input = time`, `name = log-up`, and `label = Stood up at`
 - an input with `type_input = text`, `name = log-word`, `label = First Word of the Day`, and `placeholder = What did you think when you heard the Alarm`

Note that when entering and submitting data in the form in card `Diary` the data persists in the input fields even when refreshing the page. By changing the data and resubmitting, the database entry on the server is overwritten with the new content. In contrast to this, when entering and submitting data in the form in card `Log` the data disappears and the input fields are set back to its default value. In this second case the data is stored incrementally in the database and is not fed back to the input fields.

This difference is configured by checking (store incrementally) or unchecking (update in place) the `is_log` field of the style `formUserInput`.

The screenshot shows a web application interface. At the top is a dark header bar with the text 'Projekt Name' and a navigation menu with links: 'User Input', 'Menu', 'Kontakt', and 'Admin'. On the right side of the header is a 'Profil' link. Below the header, there are two main form cards. The first card is titled 'Diary' and contains two text input fields: 'My Thoughts' and 'My Actions'. Below these fields is a blue 'Submit' button. The second card is titled 'Log' and contains three input fields: 'Set my Alarm to' (a time picker), 'Stood up at' (a time picker), and 'First Word of the Day' (a text input with the placeholder 'What did you think when you heard the Alarm'). Below these fields is another blue 'Submit' button. At the bottom of the page is a light gray footer bar containing the links 'Impressum | Disclaimer | AGB'.

Figure 4.1: A demo page with user input fields and submit buttons to send the data to the server.

4.2 Fetch User Data from the Database

The complete set of user data can be downloaded via the menu *Admin/Export* by clicking on the button **Get User Data**.

However, In order to fetch user input and display it on the screen the style `showUserInput` can be used. All that needs to be done is creating a section with style `showUserInput` and set the field `source` to the `name` of the form from which the data should be displayed.

As a demonstration let's add a new section page (set *Page Type* to *Sections*) with keyword `show_user_input` and title `Show User Input` to the demo application. Remember to check the *Header Position* checkbox (to place the page in the header) in the *Create New Page* form.

Next, add the following sections hierarchically to the page `show_user_input` :

- a container with `css = my-3`
 - a card with title = `Show Diary`
 - * a `showUserInput` with `source = diary`
 - a card with title `Show Log` and `css = mt-3`
 - * a `showUserInput` with `source = log`, `label_date_time = Date`, and with `is_log` checked

Note that if the field `is_log` of the style `showUserInput` is set the data items are displayed as a table with a timestamp column and the input field names as column titles. However if `is_log` is not set the timestamp disappears and the input field names are used as row headings.

If instead of the style `formUserInput` the style `form` is used, the input data is not automatically stored in the database. In this case a custom style or component needs to be created which is handling the user data (see Chapter 6).

4.3 Conditional Content

Sometimes it might be interesting to display data depending on an input provided by the user. Such a behaviour can be achieved with the help of the style `conditionalContainer`. A section of this style simply wraps its content in an invisible container. Further, a condition can be attached to such a section which defines whether the content will be displayed or not. This condition is defined in the field `condition` of the style. Defining a condition might be a bit tricky when first encountering it because it is based on a `json` syntax and requires a bit of getting used to. It is, however, rather flexible and might be replaced with a more intuitive interface in the future.

For detailed information about the syntax for the condition field refer to the [conditionalContainer style documentation](#).

Chapter 5

User Management

In order to view, create, modify, and delete users and groups the `select`, `insert`, `update`, and `delete` access rights of **User Management** is required. By default users in the group `admin` are granted all access rights while the users in the group `experimenter` are granted all access rights safe `delete`.

5.1 Users

To manage users navigate to *Users* in the menu *Admin*. All registered users are listed in a card on the left. A user can be either `active` (can log in and visit all accessible pages), `inactive` (cannot login as long as the account is not verified), or `blocked` (cannot login until the blocked status is reversed). The cards on the left hand side provide an interfaces to block, unblock, or delete a user and allow to manage the groups a user belongs to. A user inherits all access rights of the groups the user belongs to.

5.2 Groups

To manage groups navigate to *Groups* in the menu *Admin*. All groups are listed in a card on the left. Manage the access right of each group by checking or unchecking the appropriate checkbox in the card *Group Access Rights*.

5.3 Registration of Users

There are two possibilities to register users:

1. Each user is manually created via the menu *Admin/Users* by clicking on the button **Create New User**.
2. A user can register himself/herself in the registration form by entering a valid email address and a validation code. Validation codes are generated in the menu *Admin/Users* by clicking on the button **Generate Validation Codes**. All generated codes can be exported via the menu *Admin/Export* by clicking on the button **Get Validation Codes**.

Chapter 6

Extending the WebApp

The code base is designed in a modular way such that it is easy to extend the functionality of a WebApp. Please refer to the documentation on the [demo page](https://selfhelp.psy.unibe.ch/demo/extend)¹ for more information on extending the source code:

Workflow provides a short overview on how to manage the source code and how to report back the changes made.

The Concept Behind the Code provides an overview on the code structure and introduces the vocabulary used within the code.

Evaluate User Input explains in very broad terms how user inputs can be handled programmatically.

Customize the Theme provides several approaches to change the look and feel of the web app without changing too much of the source code.

Create a Custom Style provides a list of steps to be performed in order to create a custom style.

Create a Custom Component provides a list of steps to be performed in order to create a custom component.

Create a Custom Page provides some hints and guidelines to create a custom page.

¹ <https://selfhelp.psy.unibe.ch/demo/extend>