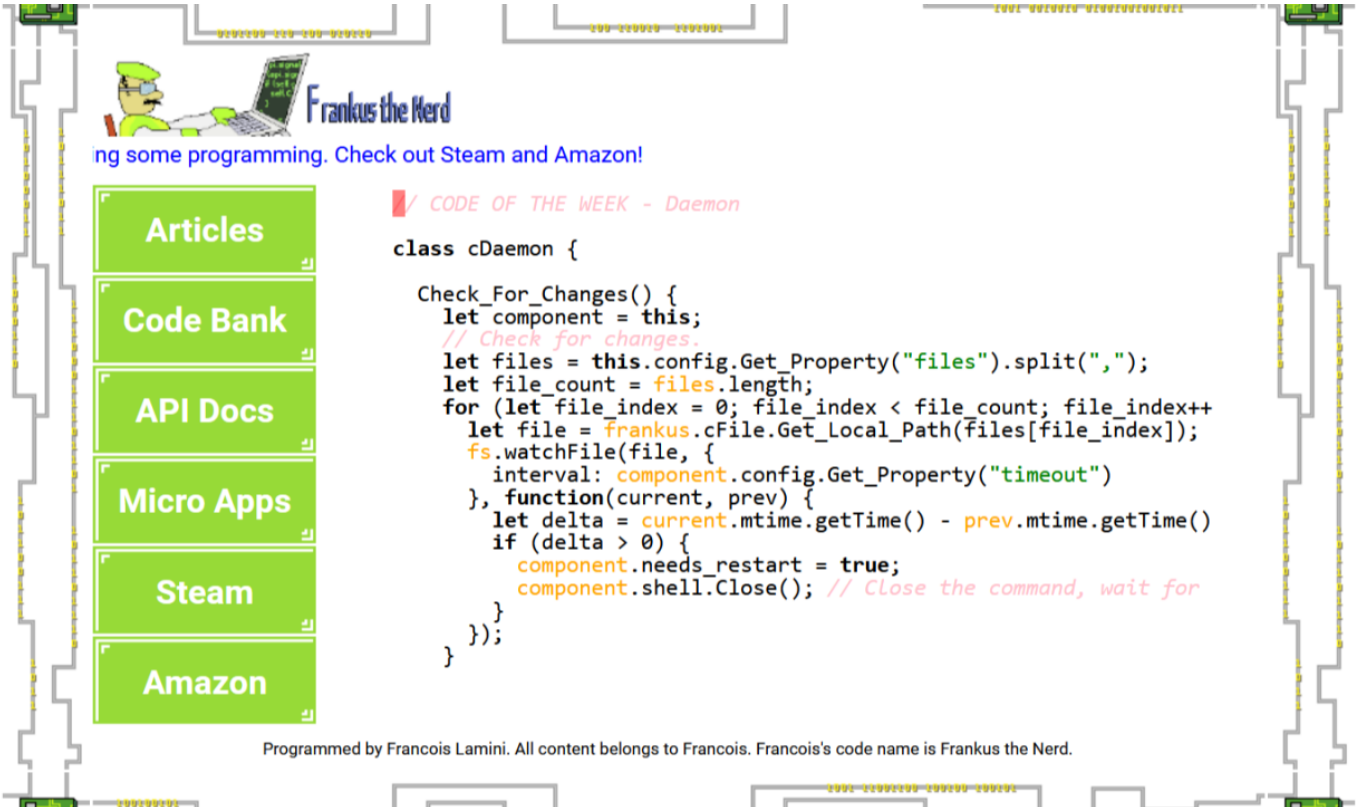


Building on the Web Using Frankus

So now it's time to build a Frankus site. We will be using Frankus, of course. Let's skip the small talk and just dive in.

Setting Up the Server

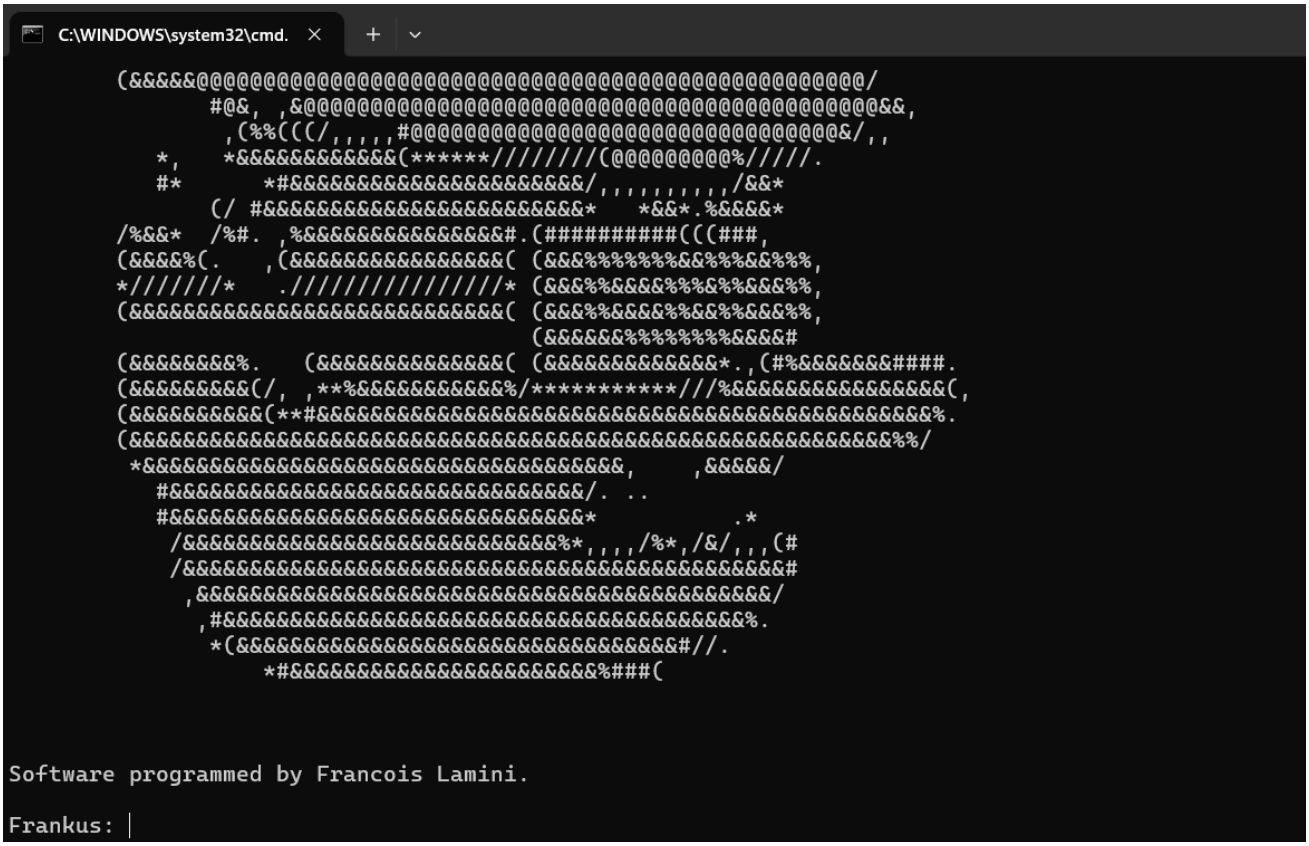
So the first thing to know is that the Frankus framework is a full stack development framework - not just a front end framework. Well, let's see how to set up the server...

First, we need to download the Frankus framework. Download it here: https://github.com/frankusthenerd/Frankus/tree/main

So, after checking out the Frankus framework go ahead and install Node.js. You can get that here: https://nodejs.org/en/download/package-manager

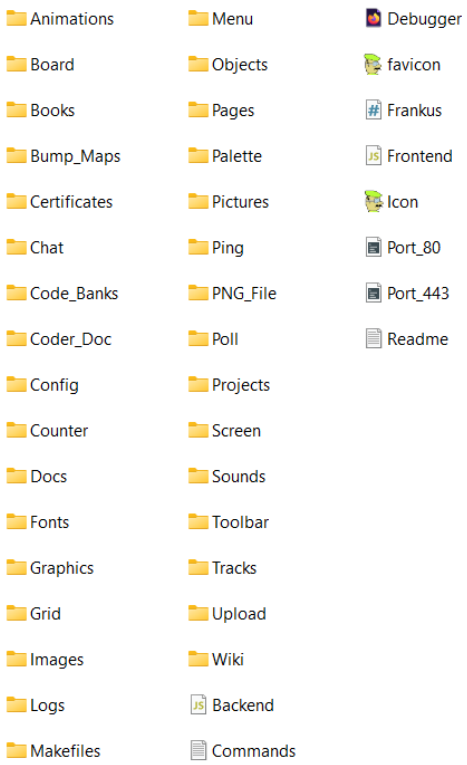
Ok, make sure that everything works by going into the Frankus_Framework folder and typing in:

It should give you the Frankus command line terminal which should look something like this:

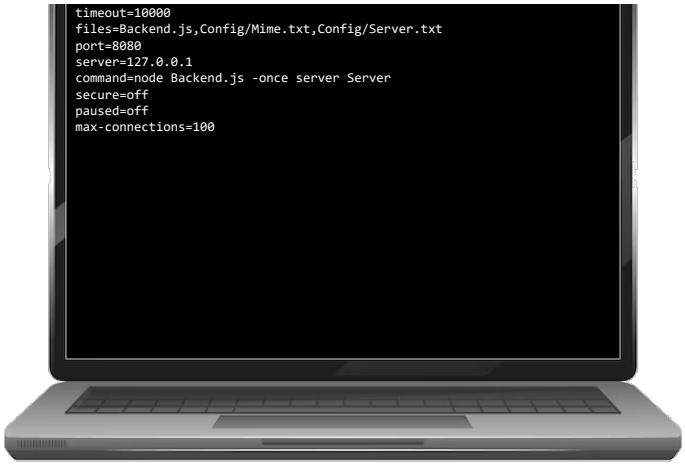


What you are seeing here is the Frankus terminal with an ASCII art image of Frankus the Nerd himself. Yeah, he rides a bike and programs on a laptop from the bike! Anyways, let's continue...

First thing we'll need is to browse the Frankus framework folder. So, let's look.



Yeah, there's a lot in here. The large number of folders is because Frankus does not store data in a conventional database - it stores files. Each component that can store data has it's own folder. Frankus can also do other things as well, such as compile C++ code, well, it the help of a C++ compiler. Anyways, we need to focus on the "Config" folder. Go in there and create a new config file called "Server.txt".



Now you might be looking at this and thinking, what the heck? Yeah, it can be difficult to understand at first but I'll break it down for you.

1. The "timeout" property is used for the daemon which restarts the server if any of the files in the "files" property are changed.
2. The "files" property contains the list of files in their respective folders that will trigger a server reload if changed on disk.
3. The "port" property is the port that the server is listening to.
4. The "server" property is the address of the server. We'll use local for now. (127.0.0.1)
5. The "command" property is the start command for the server. You call "Backend.js" with the switch "-once" to enable single command line mode instead of interactive mode. The interactive mode gives you the Frankus command line. Pass in "server" and then the name of the config file. (Leave out the ".txt" extension.)
6. The "secure" property tells the server to be on "http" if set to "off" or "https" if set to "on". For HTTPS you will nerd to have a certificate. We won't get into that for now.
7. The "paused" property is to pause the server. Sometimes you'll want to do this.
8. The "max-connections" property tells Frankus how many concurrent connections can occur on the server. Use it to throttle connections.

So, after creating your config file you can just run the server.

Type this in the Frankus terminal. It should work!

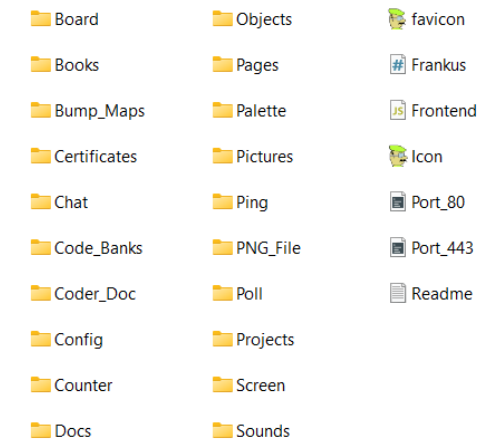
However, if you want to run the daemon type:

Oh, one note on the "Frankus:". Don't type in that. That's just the prompt on the terminal.

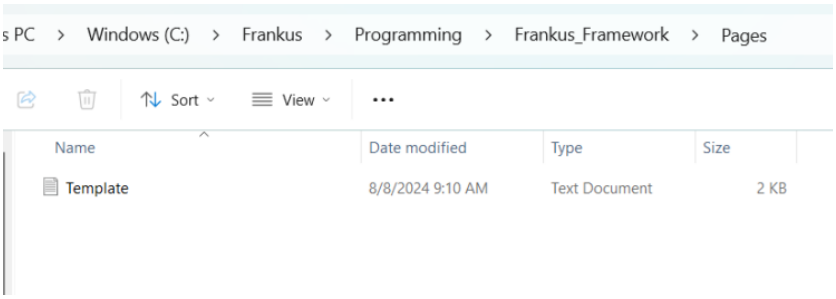
Now that our Frankus server is working we can start creating a page.

Frankus Page Building

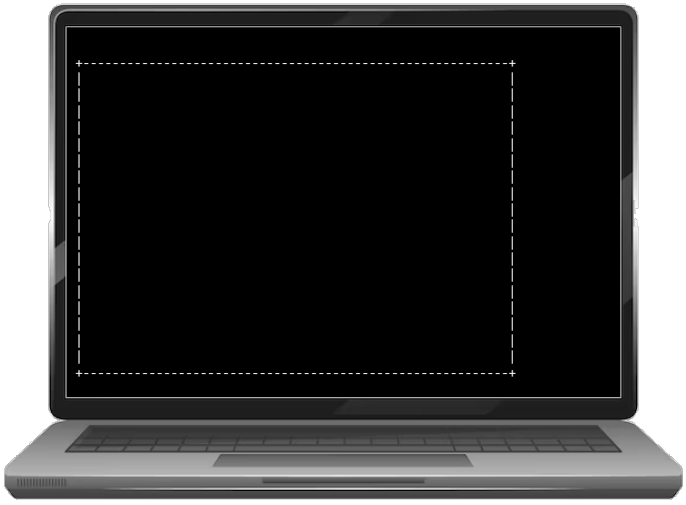
First, to create a Frankus page we'll need a layout. You could type in ASCII boxes to create the layout but if you get the size wrong (too large). It will throw and error. A better way to do this is to use a template. Open your "Pages" folder and copy the template.



Ok, copy the template.

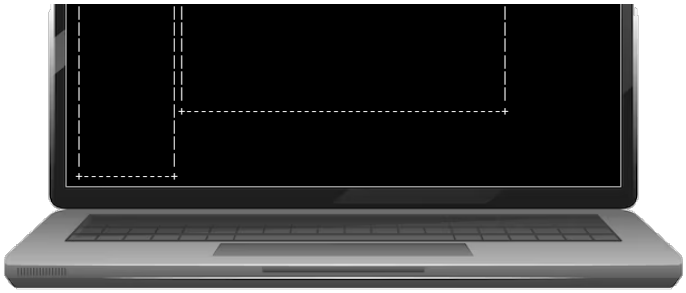


Name the new page "First_Page.txt" keep it in the "Pages" folder.

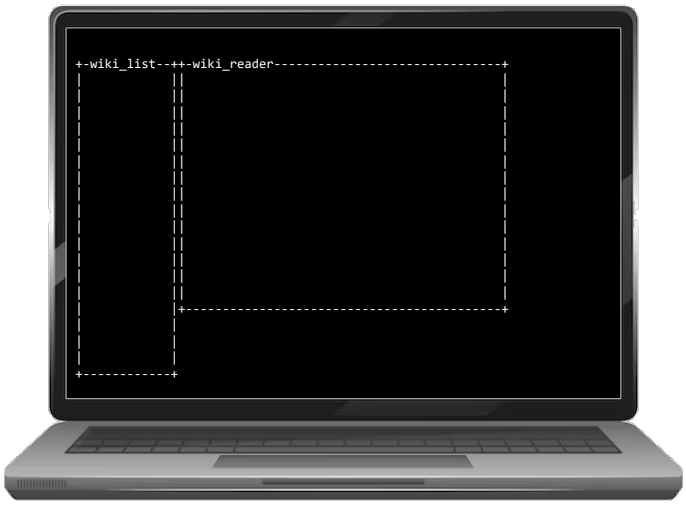


We have a box. Well, an ASCII box to be exact. So, after careful observation we see that it is 60 characters wide and 20 characters high. Each character is called a cell and those are 16x32 pixels. Please remember that. The width of the character corresponds to 16 pixels and the height corresponds to 32 pixels. When creating a page we need to pay attention to this. The page size itself is 960x640 pixels. Again, please note this. Do not exceed 60x20 characters or your site will throw an error. The Frankus parser hates that!

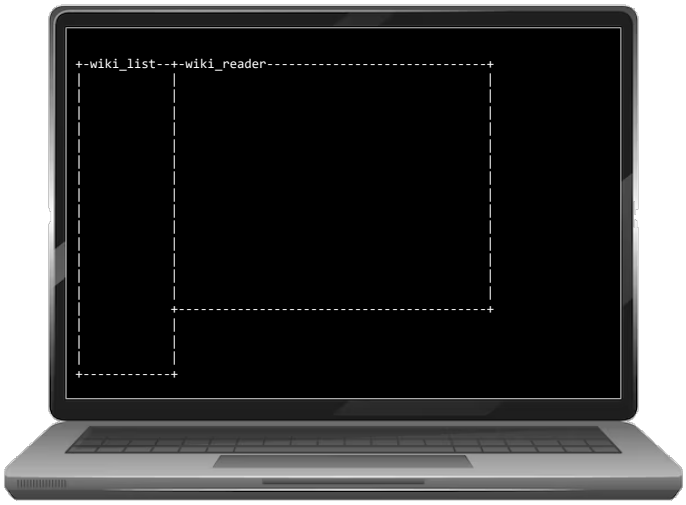
Well, let's go on. So that do we want on our page? Let's put in a couple components. First a list of articles, then a wiki reader. Ok, let's do it! Wait... now you're wondering what is the best way to edit this layout. Well, a text editor that allows overwrite mode. Anyways, let's work.



Yep, that looks good. Were you able to create the boxes easily? So let's give the boxes meaningful names.



Better... By the way, the naming of the boxes must be placed only on the top. Place the name between + and +. Don't worry about where the name is place on the top of the box. Do not overwrite + or +. Also complete your boxes - do not merge them together.



Yeah, don't do this. It's confusing and won't work.

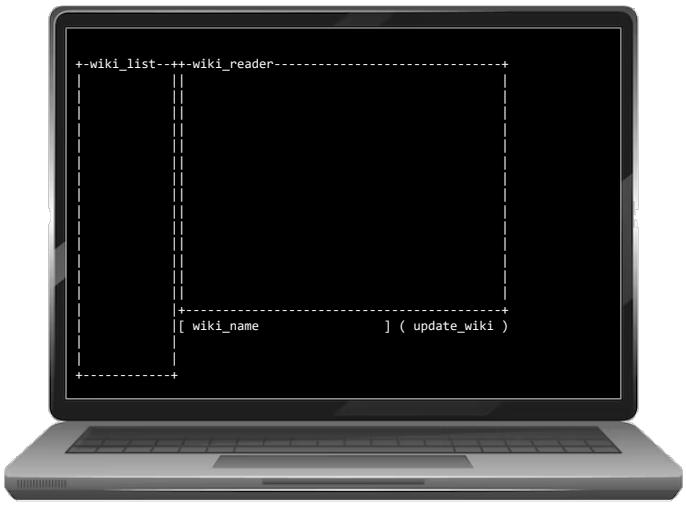
Make sure your identifiers use only characters "A-Z a-z 0-9 or _", nothing more!

Let's keep going. Let's add a field to take in the name of the wiki file on the list that was clicked. Oh, and a button as well.

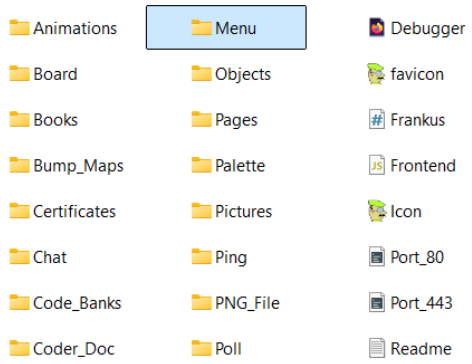
So for a field you don't need to create a box. Just use "[]". For a button use "( )". Place your identifier in the middle of that and fill with white space, like above.

So, now comes the big question. What if my identifier is too long and can that affect the size of the component if you are not careful? Yes, of course, for very small items we would be reduced to a letter but that's the price to pay to use the Frankus framework. Like any other framework it too has limitations... at least for now...

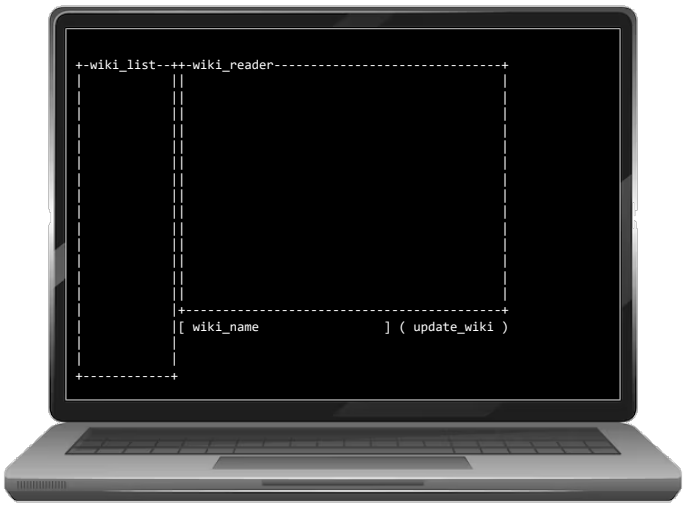
Anyways, we need to set properties. Without doing so the boxes will be boxes and will not appear on the screen - well, they'll be transparent. However, the field and the button will show up. Ok, we need to set the properties so we can override the default properties and transform the boxes into list and wiki components.



So we turned our "wiki_list" box into a"menu" which is basically a list. We want to load our wikis from a file. Don't worry about that - as we use the wiki component to create new wiki it will be added to the list. The menu itself (Wikis.txt) is stored here:



So, let's style the rest of the components.



Now you're probably wondering where in the name of J Script do you get the documentation for all of these components? You're just pulling properties out of your code box and I don't know where you're getting those from. Ok, fair enough, there are a lot of components but for now the full list of properties for Menu is as follows:

cMenu

A side menu component. Items are displayed from top down and a scrollbar is present in the menu. Each menu item is separated by a semicolon. Each item consists of a pair specifying the label and image respectively. A menu item can have either text or an image.

Properties are as follows:

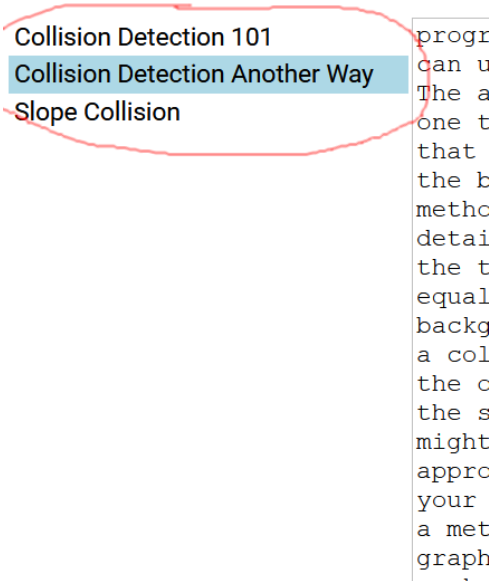
- items - The menu items with each separated by a semicolon.
- height - The height of each menu item.
- fg-color - The color of the font.
- bg-color - The background color.
- highlight-color - The color of the menu item selected.
- file - The file with the items to load.
- filter - If set to on then a filter is shown.

So before asking questions on what does all this mean? I'll explain the harder to understand properties.

items - These are the menu items. Each one is separated by a semicolon. Example:

So now you might be wonder what's with the colon and image? Well, the image is optional but each menu item can have an image by it. The image comes after the label. I. E. <label>:<image>

highlight-color - Basically, this:

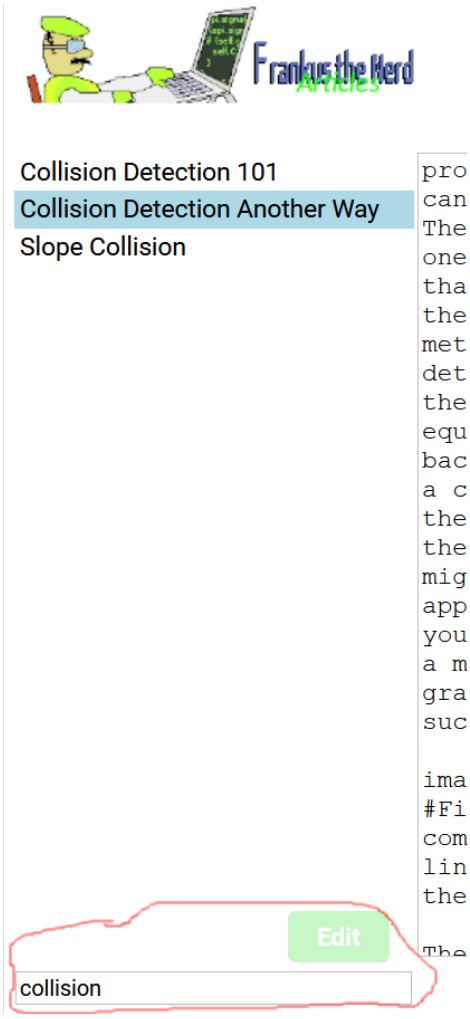


file - The file containing the menu items. The items are formatted similar to how the items property is formatted by with line breaks instead of semicolons. Example:

Uhh... what the heck? Where is the image? Well, it's optional. You don't need it. I could have also written the items property without the image. Like this:

Notice that the colon is there. Yes, it is needed. It divides the label from the image even if the image is no specified - the image would be blank.

filter - The filter. If set to "on" Then turns on a search filter. Like this:



This will allow typing of a keyword to filter out the menu items to what you want. Cool!

Wait... an edit button? Yes, the menu can be edited in real time. But why? Well, that's part of Frankus - it's part CMS! That's Content Management System for those who don't know. You can edit stuff on the client side without having to do any programming! But then can't anyone edit? Well, if the permissions are set correctly on the server, then no.

Ok, on with the next component - the wiki!

cWiki

A wiki can display and edit markdown. The markdown is sent to the server
via a passcode. The properties are as follows:

- fg-color - The text color.
- bg-color - The background color.
- border - The border around the display.
- font - The display font.
- size - The size of the display font.
- file - The file to load the wiki from.

This component allows you to add markdown code - well, Frankus Markdown, which is similar to Markdown but different. Frankus uses it's own engine to interpret this code, actually, a function. The Wiki component has an internal editor that you can access via the edit button.

```
@C-Lesh Programming Language@
C-Lesh is somewhat of a low level language with some high level features
as in languages like C++ and Java. The language is completely
structureless and uses labels like in assembly. The code, however, is
not necessarily hard to follow.

$The Memory$
Memory in C-Lesh is fixed which means that there is no garbage
collection
or any need to allocate memory as needed. Like working in assembly you
just define area of memory that you need and what's more this memory is
mixed with code! But wait a second... C-Lesh memory is divided into
blocks, not bytes, and is block addressable. A block can contain code
or data. This means that a block can be executed or treated as a no
operation. Let's take a look at the structure of a block below.

%var block = {
  code: 0,
  expressions: [],
  conditional: [],
  fields: {},
  value: {
    string: "",
    number: 0,
    type: this.code_table['n']
  },
  strings: []
};
```

Cancel

Save

So you can see that it's implemented with an HTML <textarea> as you can see the spelling error highlighting. Here you can type in Frankus markdown then hit "Save" to save the markdown or "Cancel" to abort changes.

Before we continue let's go through some Frankus Markdown.

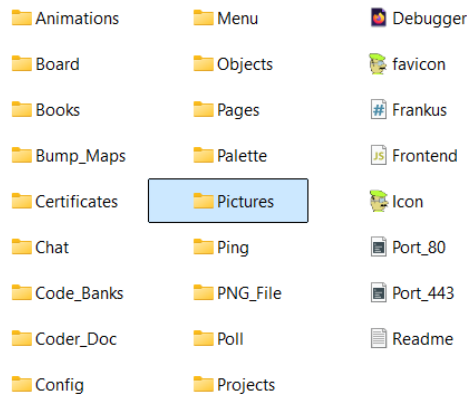
So first, how to add a heading.

Ok, good. Now what about a subheading?

Great, what about some bold and italics text?

An image, please?

By the way, images are stored in here:



Images are pictures.

What about an uploaded picture like for a message board?

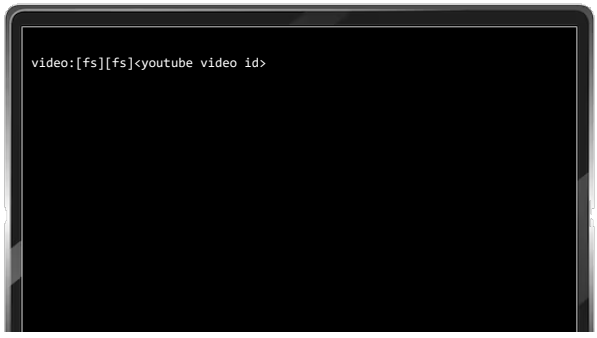
Well, easy:

This pulls the image from the Upload folder. The "image:[fs][fs]" protocol pulls the image from the "Pictures" folder.

Now let's set up some hyperlinks (http or https).

What about a progress bar to showcase the progress on your project or something like that?

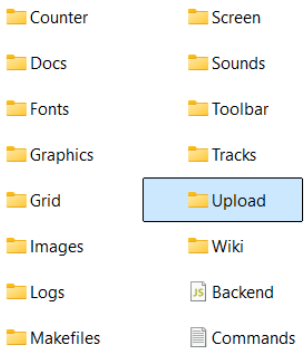
What about playing YouTube videos?



And downloads?

To place a download link:

Downloads are placed in the Upload folder, ironically!



So, that's it for Frankus Markdown... well, no. What about code?

For that just put the code markers:

Don't worry, it's formatted and space preserved.

Finally, let's place a table. We might need a table.

The "^" places a table header column. The "|" places a table data column. Don't worry about the width - they cells are all fixed width.

Last, but not least, a comment... in markdown? Yes, you can have comments in Frankus Markdown. To place a comment:

Just use the tick marks. It won't appear in the output of the Wiki or get processed.

Still there? Now let's get back to the Wiki with our new Frankus Markdown knowledge.

For the wiki, we won't set any properties because we'll load it dynamically when the wiki list is clicked.

So let's summarize we have:

- A menu component to represent the list of wikis.

- A wiki reader to display and edit the wikis.

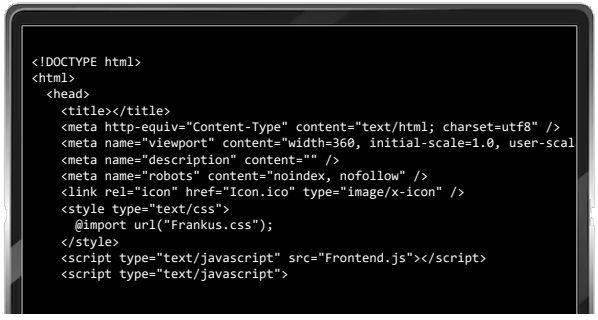
- A button to update the wiki and wiki list.

- A field to display and enter the name of the wiki.

So what now? Well, it's time to write some code!

Coding the Frankus Site

So now we need to set up the HTML file which will contain our site. Save it in **First_Site.html**.

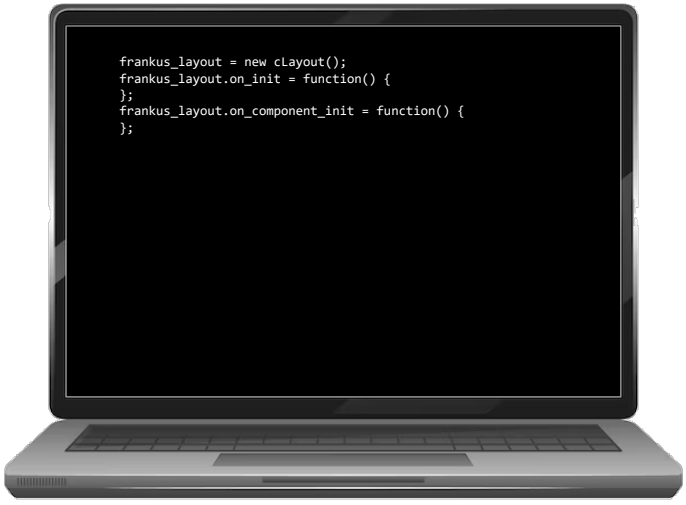


Here we go! Keep this as a template so you can create new Frankus sites on the fly! Next let's set the title.

And our description.

Notice that the body tag is empty. That's fine! Frankus will fill it in! Let Frankus do the work so you don't have to!

Now for the JavaScript. Yes, we will code that in our HTML file... or if you choose in it's own JavaScript file.



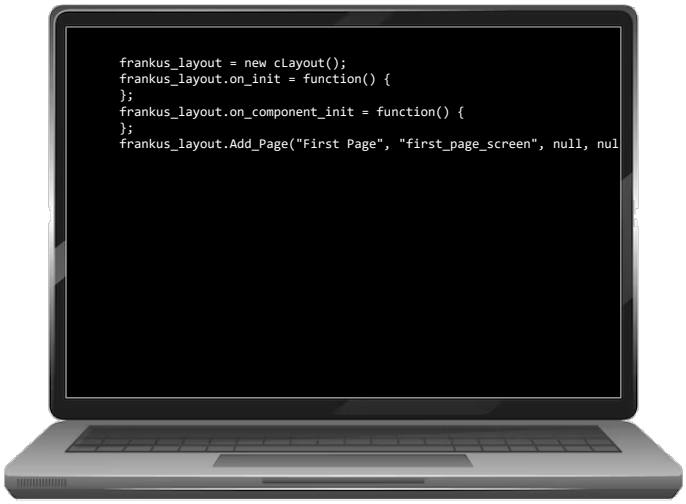
First we'll create the layout and assign it to the global variable "frankus_layout". This is a must! Do not assign it to any other variable as the layout variable to reference other components.

Next, we'll set our "on_init()" callback which fires before any component is instantiated.

Then comes "on_component_init()" which is where we write our component callback code. This fires after the components are instantiated.

Right now, save this as a template! It will make things easier in the future!

So now that we have our layout setup let's add our page.



So what did we do here? First let's look at the definition for Add_Page().

1. So the first parameter is the name of the page. This is the name that we will refer to when we open the page.

2. The name of the container that will be created for the page. Name is what you want but for convention add "_screen" to the name and keep in lower case.

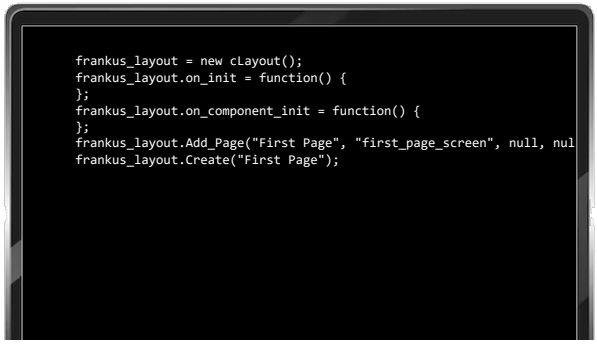
3. This is a convenient callback for when the page is paused. Some components like the marquee should be paused when the page is paused.

4. This is a callback for when the page is resumed. This happens when the page is navigated back to.

5. Remember the name of our layout file (First_Page.txt). It resides in the folder "Pages".

6. If we have a mobile version of this page it should be place in another text file called "First_Page_Mobile.txt". We won't go into that here.

For unused parameters we can pass in "null". Next, we want to create all of the pages we added as well as navigate to our default page. Easy:



Don't be confused by the "Create()" function. It creates all of the pages. The parameter passed is the page to navigate to. This is the home page.

So let's write some actual code to make the page work.

So what do we need to do?

1. We need to navigate to a wiki when we click a wiki list item.

2. We need to be able to create a wiki page and add it's name to the list.

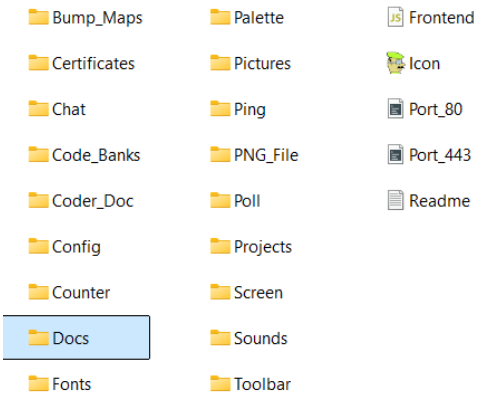
3. We need to be able to edit and update the wiki.

Ok, let's write the code...

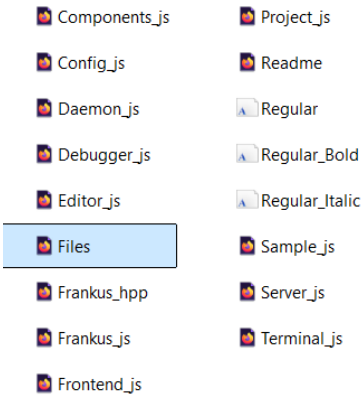
An "on_click()" handler is in order. What is passed in is the reference of the clicked component and the JavaScript event object.

We need to add the wiki if it is not on the list. How do we do that? First let's consult the documentation for "wiki", "field", and "menu" components.

Let's open the "Docs" folder in the framework.



Right here.



Click "Files.html". It will open the API.

API File List

Backend.js
Frontend.js
Readme.txt
Sample.js

Click "Frontend.js".

cMenu

A side menu component. Items are c
present in the menu. Each menu iten
consists of a pair specifying the labe
can have either text or an image.

Properties are as follows:

- items - The menu items with each s
- height - The height of each menu ite
- fg-color - The color of the font.
- bg-color - The background color.
- highlight-color - The color of the me
- file - The file with the items to load.
- filter - If set to on then a filter is sho

constructor(entity, setting
Create()
Load_Menu(items, contain

Internal routine to load the menu. It v
items All items to be loaded as an ar
container The container to load the r

Search_Menu(search)

Searches a menu and returns only th

Navigate to "cMenu". Why here? We need to be able to add a menu item.

Add_Item(item)

Adds an item to the menu.
item The item to add in menu format.

There we go! Let's code it!

Ok, pretty strait forward. So, some things you didn't know but can look up in the docs. First you can get and set a value for a field. Here are the methods:

Get_Value()

Gets the value from a field.
returns The field value.

Set_Value(value)

Sets a field value.
value The value of the field to set.

The other thing is that we can load the wiki even with a blank file. The file can be loaded with "Load_External()" function. Avoid using "Load()". You'll have to pass in the ".txt" extension. "Load()" is supposed to be for loading other file extensions. Here are the docs on that:

Load(file)

Loads the wiki from a file and displays the contents.
file The file to load the wiki from.

Set_File(file)

Sets the file to save to.
file The file to save to.

Load_External(name)

Loads a wiki document from an external file.
name The name of the file to load from.

Go with the third function unless you want to load wiki files with other extensions.

Ok, let's write the code for loading a wiki when you click on the wiki list.

So, for this one we use "component.sel_text" which is a property of the wiki component. Since the code is open source and I do insist that it should be open source. We can look in the

code for this property.

```
class cMenu extends cComponent {

 constructor(entity, settings, container) {
  super(entity, settings, container);
  this.item_selected = "";
  this.handler = null;
  this.sel_text = "";
  this.items = [];
  this.timer = null;
  this.Create();
 }
```

All properties are public - that's a Frankus thing. Whether it is right or wrong is left up to the developers that read the code and I do insist that you look at the code!

We use "sel_text" to get the name of the selected menu item. It just returns the name - that's all. After that we load the wiki. But something is wrong... we got a big in our code. Can you spot it? Maybe you have. I know programmers are smart so you'll catch it.

Do you see it? Yes, it's "update_wiki". So the problem is that it just adds a wiki page. We need it to update the name of the wiki page too. So how do we do that? So to do it we'll use the selected text of the wiki list. That value is set while a menu item is highlighted. Let's do it!

Ok, so we need to update a menu item. We will not be renaming the wiki file. To do that we call "Get_Selected_Index()" on menu. That will return the index of the selected item on the menu or "NO_VALUE_FOUND" if nothing is found. From there we update the menu item with a new name from "wiki_name" but in the process we'll lose the wiki page. We could rename the file... But for this example we'll simple allow for generation of a new file. Why? Well, rename hasn't been implemented yet and I'm not afraid to say it. Would you like to take a crack at it? Just kidding. I'll do it when I get around to it.

Anyways, as for the functions I used here is the documentation:

Get_Selected_Index()

Gets the index of the selected item.
returns The index of the selected item.

Clear()

Clears out the menu.

Update_Item(value, index)

Updates an item by index.
value The menu item value.
index The item index.

Yes, and you can clear out the menu! Seriously, if you feel like hacking Frankus do it!

```
28
29  class cFile {
30
31    /**
32     * Creates a file module.
33     * @param name The name of the file.
34     */
35    constructor(name) {
36      this.file = name;
37      this.lines = [];
38      this.data = "";
39      this.message = "";
40      this.error = "";
41      this.on_read = null;
42      this.on_write = null;
43      this.on_not_found = null;
44      this.on_denied = null;
45      this.pointer = 0;
46    }
47
```

Take a look at cFile's static functions. There is not rename function but on the server there should be.

Also check out cFile, cServer, and cE_Services in Backend.js.

Ok, that's it for now. Let's access the site.

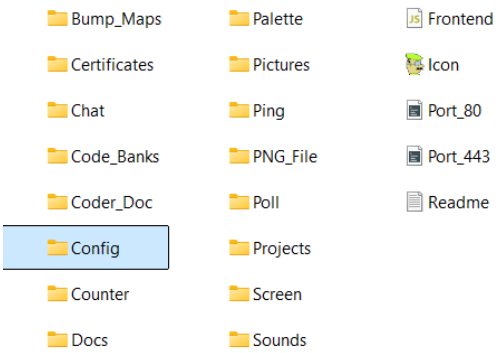
Accessing the Site

Run your app on "localhost:8080". To access our page try typing in "http://localhost:8080/First_Site.html".

We can access our page from here: http://localhost:8080/First_Site.html#First Page

So now you might be wondering if you can set up some kind of routing so you don't see the HTML page. You want it to just not show. The answer is yes!

To start out let's navigate to the "Config" folder.



Inside that folder create a file called "Rewrite_Rules.txt". Hmm... sounds like something from the Apache Web Server for those of you who have used it. Yes, this is similar to a degree...

Let type in the rewrite rules.

That's it! This mean redirect the "/" to our HTML file. The request URL is considered but without the "/". That's why the "/" is omitted and we have a blank text before to "->".

Let's do another one for a non-existent site.

Ok, so now we need to access our site along with the page. For that type in: http://localhost:8080/#First Page

That's it!

Closing Words

So this is a primer for the Frankus framework but only the beginning. Frankus can do so much more! It can also be used for command line tasks as well.