**1 - What is page, block, handler?**

<https://drive.google.com/file/d/1IJZ_UYBcN0zWx0n9avMF7FXOVkc2GPKF/view>

**Page** - Page are like our HTML Pages and this has lot of sub components.

A page should have a name & you can give url that to be displayed.

You can have JS code or CSS to your page which is global to this page.

If your page needs a json we can include in the page.

**Blocks** - A block won’t change for page to page.

Very similar to a page Example: Menu bars or Nav bars

You can have JS code or CSS to block.

**Handler** - When your writing a front-end app you can track with cookies, ajax calls, evaluate the form, this is a drag and drop.

If you want to have validation you can validate your data on failure, you can create an alert message to display user using alert and on success, you can make ajax call without writing any code and if you want to write the code you have js code where you can write your js code.

**2 - Sample block and page (HelloWorld)**

<https://drive.google.com/file/d/1ZmPg7Rb3ma_hUS7OzqiDuBbHYsY73rMf/view>

**Task-** Create first app.

**Explanation-** To create any app we need minimum a **page** and a **block**(if needed) for the frontend. We create a block named “Sidebar” with no JS code to it and some css. We can now import this created block in the page. We create a page with url “/hello-world” and include the block which we created i.e(**<sidebar/>**). No JS, CSS and JSON provided to this page. At last we simply preview that page and we get our sidebar on the left hand side.

**3 - Page Options (CSS - HelloWorldCSS)**

<https://drive.google.com/file/d/1c2zl-s_sW9XGvpRUJbh-8XlfyDQu5QHc/view>

Here the CSS is done at page level here you can make this css scoped or restricted to a particular page.

**4 - Page Route Params in a Page**

<https://drive.google.com/file/d/1HItSlRZA9817rgO-CIDEykUW8Qn7h0_2/view>

**Task-** Customize URL and make it more Generic

**Explanation-** You can add generic with “**:entitiy**” . For example we want a contact id which we want to attack to our appup dynamically, so the expected url path which we need is **/hello-generic/:id** and in our template we are going to use “**{{$route.params.id}}**”. We can add more params like entity,id. This will be very useful in creating a **single page apps** where we are reusing similar template and a page layout for a particular entity. We can add as many params as we want. To access any single param we can use “{{$route.param.**\*name-of-param\***}}”. Now let’s move on how to create a form.

**4 - HelloWorld Form (Form Fields & Handler Example)**

<https://drive.google.com/file/d/1ts2kdRXel9NPXeQ1fYOj8lFIUFAS5ArN/view>

**Task-** Create a simple form.

**Explanation-** We will create a simple form with the route **/hello/form** with no JS and no CSS. We created a basic form layout in html and we will call a **component** inside it. There are many components provided by appup, right now we are using a component for which is responsible for the creation of form by using appup. **<appup-form/>** is the tag we need to use. We need **handlers** and **params** to make the component work. Here in our case we using entity as login-helloworld. Here entity is basically in **JSON** format which we have to pass and specify. We have lot of key and value pairs for example : **fields** , **buttons** etc. So the form is generated by the type of fields described and the type of buttons we mentioned. Appup will do most of validations on itself. So basically we need to have the two main things **handler** and **workflow\_param**. We have used **pre\_load\_entity** as handler and what preload doing is, all the JSON we passed is available to us as **UIJSON**.In handler we have specified to go to particular workflow\_params we defined where we will search for fields and buttons and sending it to **uiobject**.

Here is the sample code from the handler for assigning the fields as well as buttons respectively.

* ui.formFields=window.UIJSON.[ui.workflow\_params.entity].fields;
* ui.buttons = window.UIJSON.[ui.workflow\_params.entity].buttons;

**5 - Form Properties (Classes, Computational, Conditional, Dynamic Dropdowns etc)**

<https://drive.google.com/file/d/1ezGuoJ2fw_8QoeTsA5HjqV21qJxzYB6e/view>

Here we are creating a contact form with no attributes.

We are using **preload\_entity** for loading handler & we have a json to create buttons,fields dynamically .

In our JSON we are having different fields:

**name**: that you want to see in deserialized,

**label** : To display

**description**: to display down the field,

**type**:to give the type of field text/button/dropdown etc.

**class**: that you want to bind to that field.

**id**: is like normal html id which should be unique.

**url**: to display the route params that you want to show in your url & you can also get json data as an input that to be displayed in your page.

**computational** : will update the present url with the url that your provided in url field

**options**: that have to be displayed in the dropdown box

**required** : is like a condition which checks for true value and evaluate the given input.

**key\_label :** that to be displayed

**Key\_value:** like a value here to store value of the option.

**buttons** : is a array you can pass object in side that like label,function,type,variant

**post:** is like route which will take you to the next page on success.

**server:** where the data has to been go.

**nextURL** : is like once the post is completed it will take you to the next url

**Code:**{

"contacts": {

"fields": [{

"name": "name",

"label": "Name",

"description": "Please enter your name",

"placeholder": "Enter your name",

"id": "first\_name",

"type": "wysiwyg",

"required": true,

"class": "col-sm-12 col-md-6"

}, {

"id": 10546,

"type": "text",

"name": "URL",

"label": "URL",

"required": true,

"class": "col-sm-12 col-md-6"

},

{

"id": 1090,

"type": "text",

"label": "Friendly URL",

"name": "url2",

"required": true,

"computation": "'https://'+form.URL.replace(/[^a-z0-9]/gi,'').toLowerCase()+'.appup.cloud'",

"disabled": true,

"class": "col-sm-12 col-md-6"

},

{

"id": 10,

"type": "dropdown",

"name": "cloud\_type",

"label": "Cloud Type",

"option": [{

"value": null,

"label": "Please select an option"

},

{

"value": "Appup",

"label": "Appup"

},

{

"value": "Amazon",

"label": "Amazon"

},

{

"value": "Google",

"label": "Google"

}

],

"required": true,

"class": "col-sm-12 col-md-6 select"

},

{

"id": 106573,

"type": "text",

"name": "SERVER\_URL",

"label": "Enter Server URL",

"required": true,

"condition": "form.cloud\_type!=null && this.form.cloud\_type=='Appup'",

"class": "col-sm-12 col-md-6 red"

},

{

"id": 10,

"type": "dropdown",

"name": "name",

"label": "Select Name",

"key\_value": "id",

"key\_label": "name",

"url": "https://jsonplaceholder.typicode.com/users/",

"required": true,

"class": "col-sm-12 col-md-6 select"

},

{

"id": 50,

"type": "checkbox",

"name": "checkbox",

"label": "Checkbox",

"option": [

{

"label": "Pineapple",

"value": "pineapple"

},

{

"label": "Grape",

"value": "grape",

"disabled": true

}

],

"required": true

}

],

"buttons": [{

"label": "Submit",

"variant": "primary",

"type": "submit",

"function": "onSubmit"

},{

"label": "Submit",

"variant": "primary",

"type": "submit",

"function": "onSubmit"

}],

"post": "https://our.appup.com/login"

}

}

**6 - Generic Form - Routes - Page Templating (:entity)**

<https://drive.google.com/file/d/1NuxFeaJNG6YVKifP0Agpkw22jUbOuw9K/view>

**Task-** Create multiple form with ease.

**Explanation-** As we now know how to create a form. Now if we want to create multiple forms with same page layout we can do that by passing the whole form fields as **JSON** object. For eg: in the video we have used contacts2 and login. And in the handlers we will now use another handler **“pre\_load”** simply pass **route.params.entity.**

Here is the sample code from the handler for assigning the fields as well as buttons respectively.

* ui.formFields=window.UIJSON.{{route.params.entity}}.fields;
* ui.buttons = window.UIJSON.{{route.params.entity}}.buttons;

And for the workflow\_params we will pass it as

<appup-form preload="pre\_load\_entity" :workflow\_params='{entity:`${$route.params.entity}`} />

**7 - Form Submit**

<https://drive.google.com/file/d/1Ng2GtPhS1pmiYPvOleNnxsMo0Bz6DYbA/view>

When ever we create a form and validate it and send to backend server we create a handler and in json we have given the post where it needs to post, when the data is validated it is going to **submit handler,**

It first validate the form and if it is success it will make a ajax call for this we need to add the server

**{{ui\_json.server}}/{{#with (lookup ui\_json route.params.entity)}}{{post}}{{/with}}**

**ui\_json.server** : will search for server in json,

**lookup:** will search in the ui\_json for key and try to iterate and look for post the data

This way you can make ajax calls and create a form ease.

8 - Form Edit and Put

<https://drive.google.com/file/d/1zP6d-WEON4338VxS4OemseSK0BfErlO4/view>

**Task-** Pre Populate data (use already saved data dynamically).

**Explanation-** In order to perform this we need to use another handler now which we call as “**pre\_load\_edit**” which will append to the **GET** request and get the data, **deserialize** it to the form and when someone clicks on **submit** button then it will **serialize** the data. As we seen in the video we get the data from url defining in handler as “{{ui\_json.server}}/{{#with (lookup ui\_json route.params.entity)}}{{get}}{{/with}}/{{**route.params.id**}}”, we can notice that in the end we checking for **id.** We set a variable named as response\_data to take the output and again we include a JS code and we set a new variable **data** and assigning the output variable (response\_data) values to this data variable. Here is the code for further reference :

**CODE-**

function(data, ui)

{

data.response\_data = data.response\_data.data;

return data;

}

And then this code will be further sent to deserialize. And as mentioned before we want to serialize it when clicked on submit, so we passed it to the handler “**put\_serialize**” where we simply validate and do the **AJAX** call as **PUT** method and the rest is same like before.

**9 - Functions in Pages**

<https://drive.google.com/file/d/1DAlCxQ2SSxp6jAu_BUnEuxFemEBm6sc2/view>

If you want to add some additional functionality to our pages

Imagine a page where you want to create a title dynamically or a costume function like you want to show alert or get data we look at two types of functionality one is **computed** and other is **methods**.

**computed**: We are creating a new props call **title** and the title gets call and returning test you can use it in your page like {{title}}

**methods**: for context which is called when you click, it is like when your clicking it is call is a function called test and getting an alert message.

**10- Final Generic Form with preload, submit, back, title**

<https://drive.google.com/file/d/1R2LmABR93CnauQhen8l3JY_JoLyxHZu3/view>

**Task-** Finish Up and finalize form with function.

**Explanation-** If we want to add **custom** title and custom href link we can take use of **functions**. We will use **computed** property, and inside that block we will define **title** and **href** and this title and href will be passed by in **JSON** format. Here is the code for further reference :

**CODE-**

computed: {

title: function () {

return window.UIJSON[this.$route.params.entity].title

},

back: function() {

return window.UIJSON[this.$route.params.entity].back

}

}

After using those computed fields our code should look like this :

**CODE-**

<div class="float-left">

<button type="button" class="btn btn-outline-secondary bg-white" :href="back">Back

</button>

<h1>New {{title}}</h1>

</div>

Here href is binded with back which is a computed property. Remember to add **colon(:)** or else it won’t work as it won’t bind dynamically.

Also make a serious **note** that **Generic** types should never be allowed to passed on with any conditions like if title has some value or anything, because generics are meant for general purpose and if we use any conditions on to it, then it become a **non-generic**.

If we want to add some extra functions just write them in JS in a method block for Eg:

**CODE-**

methods: {

test: function(){

alert('Button is clicked')

}

}

**11 - Collection - Table (also v-for)**

<https://drive.google.com/file/d/1AFKTGqI6jYeQYFinsafj5NiCmlzUf8Xx/view>

Collection is a set of items, we create the json that the way we want to show the information that to the user.

We have <appup-card> and in side the handler we write **ui.sortKey=window.UIJSON.{{route.params.entity}}.sortKey**

It sets the sortkey for example like based on name,location so on. In our json we dont have any sortKey if you want we can have it. After that it make a ajax call, it is a get call

function(data, ui)

{

ui.data = (data.response\_data.data)

}

ui.data is a appup-card prop and we are setting it with the response that we got in the ajax call.

If you want to display the data that we brought from ajax is show in the json format which is not user friendly

We have created a slot and we can show only a particular data that to be displayed to the user using **v-for** loop on the data that we brought from ajax call.

<https://drive.google.com/file/d/1-QIiRbtFmbipYY4ZIYV8wgvfV0AA0ui2/view>

filter: this is not a client-side, if you had seen the database tutorial for backend, when your generating the backend call and you can also request to get only the required data it is mandatory when your having your own backend.

**12- Collection - Card (also v-if)**

<https://drive.google.com/file/d/1vUrxLEcmflgUJsdxil_T-eNuIvB-Kk_q/view>

**Task-** Implementation of Appup Card

**Explanation-** Like before we will use the same handler as “**preload\_grid**”. We came to know about a conditional statement **v-if** to perform some conditions. It’s simple if condition like other programming languages.

**CODE-**

<template v-if=”item.username != ‘Samantha’”>

{{item.username}}

</template>

<template v-else>

<p>Hi Samantha</p>

</template>

Here we have to also use slot and pass “**section**” to it. To show it as card or table like before.

**13- Collection - Card with Sort Key**

<https://drive.google.com/file/d/1xA9GfqFjQbjC_m_-iRyK72eVvMQYhHXM/view>

We can put a particular sortKey called “username” we are saying that use this particular name to sort the user data, when you sort the data which we are getting is split in junks based on sortKey.In the sorted\_data we got key value pairs.

**14- Collection - Card with Search (Events)**

<https://drive.google.com/file/d/10hkDstUVNrhb5eue6iX_I9F7FPbDRz_B/view>

**Task-** Add more functionality to card i.e searching

**Explanation-** Now in order to perform search operation we need to add one more element that is the **input** element for searching. Like discussed @**click** event we will use @**keyup** event in this case. So whenever this event is raised we **trigger** it, here trigger is like a function **call** given by **appup** to you. So when someone press any key @**keyup** event will raise, which in the end is calling a trigger which will be referred by some component, and the next value will define what type of operation is to be done (**search**,**delete** etc). **search** is a build in event provided to you by **appup**. Then the next thing needed is the **parameters** that is the value. And at last you need the **unique** **key**. So if we use key then we increase the efficiency by **restricting** the data to be searched.Here is the code sample to for further reference:

**CODE-**

@**keyup**="\_trigger(**'triggers'**, **'search'**, `**${$event.target.value}**`, **'name'**)"

Here

* **triggers** is the reference name used in other components, in our case <appup-card/>
* **search** is the type of operation we want to do
* **${$event.target.value}** this will return us the value that we want to search
* **name** and the last to restrict the search operation we will provide unique key where the operation will be performed.

**15- Collection - Card with Execute Workflow (Delete example)**

<https://drive.google.com/file/d/1KmpQcaJPXEcWoY_fZ2bE-5KQKqskzA6s/view>

**Task:** How to trigger and even

<b-dropdown-item @click="\_trigger('triggers', 'exec', 'grid\_delete', {'key': `${item.id}`, 'entity': `${$route.params.entity}`})">Delete</b-dropdown-item>

**Explanation:**

[We have two option one is edit and other is delete.](https://drive.google.com/file/d/1KmpQcaJPXEcWoY_fZ2bE-5KQKqskzA6s/view)

If you look at delete @click we are triggering an event on the appup card and ‘exec’ is shortcut for execute handler and the name of the handler is grid\_delete, and we are sending the key and entity, this will remove the element from the frontend and find the delete key in the json and it is posted.

In handler we delete the data based on the key and setting it back to frontend.

In backend we are looking for entity and sending a delete key.on success we show an alert.

**16-Collection - Card with dynamic templates (using Template Literals)**

<https://drive.google.com/file/d/1_oqtCCch_DNbxsPFdjAwrMbUb2p0YanQ/view>

**Task-** How to include the card component dynamically.

**Explanation-** We will include the following JSON key :

**CODE-**

"card-template": "HB <h5 class='card-title'> ${item.name}

</h5> <p class='card-text '>

${item.email}

${item.username}

</p>"

Here in the above code we are using javascript literals (item.email, item.username). We also have to write a wrapper function for that, here is the code for further reference:

**CODE-**

methods: {

card: function (item) {

// Get HB Data

let cardTemplate = window.UIJSON[this.$route.params.entity]['card-template'];

// Return if not defined

if(!cardTemplate)

return;

return eval('`' + cardTemplate + '`');

}

}

Here we have a function called **card** which will get the key for **card-template** and **evaluate** at runtime. And in our template we will just call v-html and passed the value which is being returned by card method. This method will return dynamic html code.

**CODE-**

<span v-html="card(item)"> </span>