Discovering Nuxt with Form Builder and Table Component

Swarnima Joshi

Resum– La creació d'un lloc web intuïtiu i cosmopolita requereix l'assistència del marc adequat i la definició del marc adequat en el desenvolupament web depèn de l'abast i el requisit del projecte que s'està desenvolupant. Entre molts altres frameworks populars, Vue ha estat un marc JavaScript de front-end progressiu utilitzat per crear interfícies d'usuari. Aquest projecte està inspirat per entendre Nuxt, un marc de Vue, desenvolupant components basats en això i, eventualment, migrant la base de codi Vue a Nuxt. Els dos components principals que s'estan construint són el component Form-Builder i el component UserTable, per estudiar la compatibilitat i l'accessibilitat de Nuxt. El primer component permet als usuaris crear el seu propi tipus de formularis depenent del sector per al qual l'utilitzin, mentre que el segon component es va crear per mostrar dades juntament amb altres funcions com ordenar, filtrar, editar, afegir i suprimir elements d'aquesta taula.

Paraules clau– Constructor de formularis, framework, Vue Formulate, Component de la taula, Bootstrap, schema

Abstract– Creating an intuitive and cosmopolitan website requires the assistance of the right framework and the definition of the right framework in web development depends on the scope and requirement of the project being developed. Among many other popular frameworks, Vue has been a progressive front-end JavaScript framework used in building user interfaces. This project is inspired to understand Nuxt, a Vue framework, by developing components based on so, and eventually migrating the Vue code-base to Nuxt. The major two components being built are Form-Builder Component and UserTable Component, to study Nuxt's compatibility and accessibility. The former component allows users to make their own kind of forms depending on the sector they are using it for whereas the latter component was built to display data along with other features like sorting, filtering, editing, adding, and deleting items from that table.

Keywords– Form Builder, framework, Vue Formulate, Table Component, Bootstrap, schema

1 Introduction

Form builder is a tool that helps people build their own kind of forms with their chosen fields, making it highly dynamic, saving time for copy pasting to create a new form, automating the entire process of building so and simply making things easier and quicker for businesses, organisations, institutions, teachers, students and professionals to build a varieties of such documents. And one of the simplest and most effective ways of displaying data are tables. So the idea of this project was to discover Nuxt [1] by developing these two component and eventually establishing it as the major candidate for the substitution of Vue. The brief introduction of the Form-Builder component would be, it lets users of all kinds to have as many different forms as they need with as many distinct fields as per their requirement. This led to eliminate any kinds of restrictions that the built-in static form can bring. And about the table-Component, it was not only aimed at meeting the objective mentioned above but much more. For this project, SaaS [2] is being used to manage CSS [3] and heavy use of bootstrap Vue [4] is made, including libraries like Vue-Formulate [6] and more. When using components from external CSS frameworks like bootstrap [5], although it comes of with many exciting features, it might feel restricted in comparison to custom made components. So table-Component was also developed as a research to elim-

• E-mail de contacte: swarnimajoshi902@gmail.com
• Menció realitzada: Tecnologies de la Informació
• Treball tutoritzat per: Sara Estravis Nieto (Department of Information and Communications Engineering)
• Curs 2021/22

inate those restrictions by not using the bootstraps component directly, rather developing base components based on those bootstrap components. This way more fields could be introduced to your custom component and repetition of code could also be avoided.

In the following sections and subsections of the article we will be going through the objective of the project, the methodology applied and planification required for its completion,technology used, detailed explanation of development process, results obtained and conclusion .

2 OBJECTIVE

The long term objective of this project was:

- Build the form builder component which can be used across different platforms and organizations
- Build the custom made tableComponnet to display the data and replace bootstrap based table-Component.

The generic objectives to be achieved were:

- Final product had to be scalable so that it can be reused
- Better User Experience with user friendly interface
- From the developer point of view, it was imperative to follow atomic structure for code management and build reusable component

3 METHODOLOGY

Continuous improvements were required for this project development. So setting the foot forward with Scrum [7] methodology of project management. The project was divided into sprints.After each sprint, the code was reviewed, all the features were tested properly so as to remove all kind of dissatisfaction before moving forward with next sprint.

4 PLANIFICATION

Planification of this project consisted of different phases, it started with Preparation phase, then Draft phase, followed by Design, Development, Testing, Deployment and Documentation as shown in the Fig. 1 of Appendix A.

4.1 Preparation of Project

The principal phase of the planification process was to define the objective of the project, discussing what features and fields were wanted in the final product and what would be the scope of the built component. Similarly, assigning minimum and maximum time required for its completion was also done during this discussion. It also consisted of researching similar projects, understanding what was already in the company's platform, what were the missing points and what had to be achieved next. So as a result, sketching out the draft of functionality and code structure was done.

4.2 Design

After having a rough sketch of what functionalities were wanted, it was now time to prepare a clean UX design. After its completion,it was rechecked to finalise how feasible and pragmatic it was. Based on those arguments, the plan would either move to the development phase or some modification would be brought to the current design.

4.3 Development

This phase mostly consisted of coding and giving a shape to the objectives defined. It started with building smaller components and then integrating all of them to have a final product. The main focus thus, was to develop a reusable and customizable element keeping the clean code structure.

4.4 Testing

After the completion of development phase, the product was sent for testing. Each features were tested using end-end testing framework, Cypress, on different platforms and screens. Once all the test passed the plan moved to the deployment Phase. For failed test, it required going back to development phase to fix the bug.

4.5 Deployment

On the successful running of tests, it was time to release the product and have it ready for users to use it.User's response to the new product release were taken into account in order improve the UX standards.

4.6 Documentation

The final phase of planification process is formally documenting the entire process of development, recording the results obtained and officially closing the project.

The timeline of the planification is represented as a Gantt chart in figure above as Fig. 2 of Appendix A.

5 TECHNOLOGY STACK

This project is front-end based project, so the tech stack used for its development are mainly Nuxt and Saas. For the code management, Git was used as a continuous development tool.

5.1 Nuxt

Nuxt is an open source framework of Vuejs (a javascript framework) [8] that is shipped with plenty of features to boost developer productivity and the end user experience. The reasons why this framework was chosen for the project are:

- It needs zero configuration, app can be coded right away
- Automatic routing and code-splitting for every page, unlike in Vue or any other javascript framework, developer doesn't need to write code to define the routes.
- Nuxt auto imports the components built

- It allows Universal Rendering (both client side and service side) which in-turn helps in Search Engine optimization (SEO) [9].
- Easy integration with SaaS, Vuex [10] and any other Vue-libraries or modules

5.2 SaaS

SaaS is a preprocessor scripting language that is interpreted or compiled into Cascading Style Sheets(CSS). When the project gets bigger, large numbers of files are encountered and each of those files are associated with unique CSS. It is messy, unreadable to developers and hard to maintain when CSS is directly used. SaaS allows to have different folders to maintain the code structure in atomic style. Sass lets you use features that do not exist in CSS, like variables, nested rules, mixins, imports, inheritance, built-in functions, and much more. It reduces repetition of CSS and thus saves time.

6 State of the Art

Form builders have existed for a long time and have been used in different way. The working mechanism of these tools looks very simple. All that needs to be done is to create labels and questions and respondents can provide the information that is being looked for through text boxes, drop-downs, slides, description box, radio buttons, and much more. The fields could be set as required or optional and restriction can be applied to the the types of responses that is being received to have more control over the data being collected. Even though building forms and getting answers looks quite straightforward process, these apps come in all shapes and sizes depending on the requirement of the user or the business it is required for. So what makes some of them great and other not so much?

The differences are in following points:

- Simple to use: Easy understandable User Experience(UX) is the only way for the product's success
- Highly customizable: It should be highly flexible with the changes and should allow users to customize their form with wide selection of options
- Easy to distribute: The forms built needs to get to as many participant as possible
- Powerful analytical tools: The data collected needs to be analyzed and should be able to be exported

The most commonly used reference would be google forms, which allows the incorporation with google docs, google sheets or google slides. It lists all fields or questions from another Google Forms or google platforms mentioned before and makes fields available to import into the new Google Form allowing users to easily select and import necessary fields or questions into that Form. It allows easy export of Form Questions so that whenever it has to be reused, it can simply be imported back.

Besides this, there are other popular Online Form builder tool in existence, such as:

- Microsoft Form: for collecting and analyzing form results in Excel
- JotForm: for building a form from a template
- Formstack: for advanced analytical and regulated industries
- Typeform: for conversational data collection
- Paperform: for creating order forms
- Formsite: for protecting sensitive data

Tables being used as the way of representing data across multiple platforms, especially larger data set is nothing new. There exist simpler tables to highly advanced excel like tables in the market. A good table representation of data would be a cleaner, easily readable and easily usable.

While there are so many easily accessible options in the market, sometimes you still fall sort due to the specific requirement of the company which might include need of a specific field or a specific layout or even specific client preference.

7 Project Development

The base of project development is project designing, so the first thing to do was have a design ready for the product to be developed. And in order to design the product, the requirements need to be set beforehand. Below we will mention some of the functional and non-functional requirement of this project.

7.1 Requirements

As requirement process is the critical one for the success of any system or software project, we will see some of them categorised as functional and non-functional in the subsection below:

7.1.1 Functional Requirements

For Form-Builder Component

- There should be a drag (where field required for form are located) and drop zone (where field are dropped)
- The fields should be cloned once dragged and each copy must be associated with unique key value
- Each dropped element must have option to be edited and deleted
- User should be allowed to choose whether the field is required or not while still in the drop area
- User should be allowed to see the preview of the form built so far
- Forms built should be previewed.

For table-Component

- Showing skeleton loader for 2 secs before displaying the table
- A button to add new item to the table
- A column of actions to edit or delete the existing item
- Sortable columns
- Filtering options established outside the table and a button to reset already established filters if any

7.1.2 Non-Functional Requirements

For Form-Builder Component

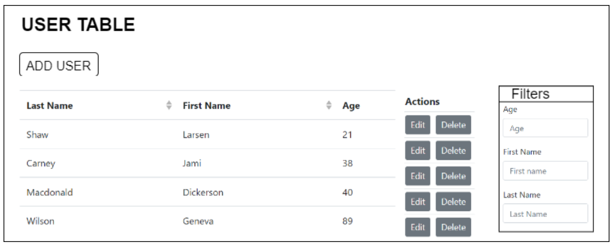
- The drop-zone and drag-zone must be clearly visible and understandable to the users
- The preview and save button must be understandable to the users.
- There should be icon next to each dropped field to let users know that the fields can be rearranged in the drop area as per their liking.
- Once the form is submitted user should return to the main page.
- The design of the product should be user-friendly.

For table-Component

- Edit button should open modal which must have the current data of that chosen row
- After submitting the forms used for adding and editing the items in table, the modal should close and show the table page
- User friendly
- Component developed should be reusable

7.2 Design

7.2.1 Design of the Table Component



USER TABLE

ADD USER

Last Name	First Name	Age	Actions
Shaw	Larsen	21	Edit Delete
Carney	Jami	38	Edit Delete
Macdonald	Dickerson	40	Edit Delete
Wilson	Geneva	89	Edit Delete

Filters

Age

Age

First Name

First name

Last Name

Last Name

Fig. 1: Draft of design of Table

Bootstraps b-table component acted as an underlying component for the design shown in (Fig.1). The objective was to make custom table components with b-table only acting as a base. This table supported features like sorting the columns and it had a separate small table for users to enter the filters, to make the search of items in the table easier. A reset button in filter table was also introduced in order to allow users to clear the filters used. The buttons being used were also customised in a sense that they were not using b-buttons of directly, rather a skeleton for all the buttons being used in the project was made and that single skeleton was used everywhere, the only difference in all those buttons being the data passed. This concept applied to modals [11] used in the project too. Basemodal was built using b-modal and any modals being used in the projects used this Basemodal, avoiding a lot of code repetition. The buttons used in Actions column of table opened up modals which were using Vue-formulate libraries to display forms to users in order to capture data entered by them.

All the operations done on the table were recorded in Vuex store with help of getters, mutations and actions. It was from where data was displayed and also saved, in case of users making any change to table.

7.2.2 Design of the Form-builder Component

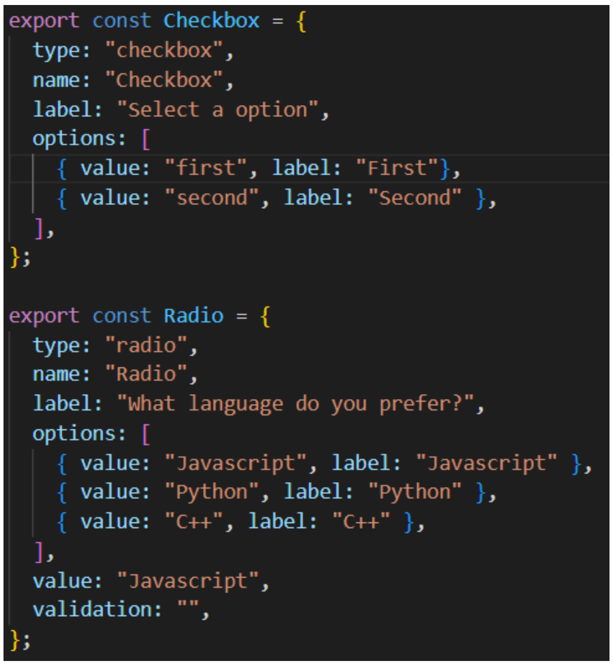
For the second component, Form-Builder, the inspiration of how the design should potentially look like was taken from Dribble [12]. However, design and development go hand in hand, sometimes due to development restrictions, designs might need slight deviations. For example, regarding the edit feature linked with each field, the initial design was made in a way such that users could modify the properties of that field right from the position they were but as the development had started, it looked better to have one edit button that allowed to do the same actions once it's clicked, so a layering was added to that action. For the better user experience, icons were added next to each field so that it was clear that they were also drag-gable between themselves.

Regarding the fields used, we have different types, some of them are:

- Text Box: input type is text
- Information section: a group of multiple input which includes a text box, Url, image and Paragraph
- Select: input type is checkbox
- Radio button: input type of radio
- Date Selector: input type of date
- Slider: of type range

Since the design was based on schemas [13](Fig. 2), it was very easy to add or remove fields as per requirements. So in that sense the design was prone to easy customization. The buttons used were based on reusable component 'BaseButton' like we discussed previously. Same goes for the 'BaseModal' which for example is being used in case of Preview and Edit-Button. Before submitting the form, it was made sure that the user input coincided with the input type of fields and to do so validations were set for each one of them with the help of Vue-formulate library.

The draft of the design can be illustrated above in Fig.3.



```
export const Checkbox = {
  type: "checkbox",
  name: "Checkbox",
  label: "Select a option",
  options: [
    { value: "first", label: "First"},
    { value: "second", label: "Second" },
  ],
};

export const Radio = {
  type: "radio",
  name: "Radio",
  label: "What language do you prefer?",
  options: [
    { value: "Javascript", label: "Javascript" },
    { value: "Python", label: "Python" },
    { value: "C++", label: "C++" },
  ],
  value: "Javascript",
  validation: "",
};
```

Fig. 2: Schema of Fields

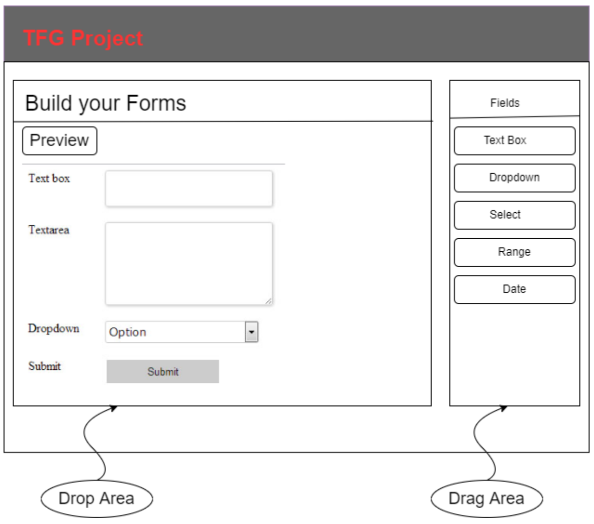
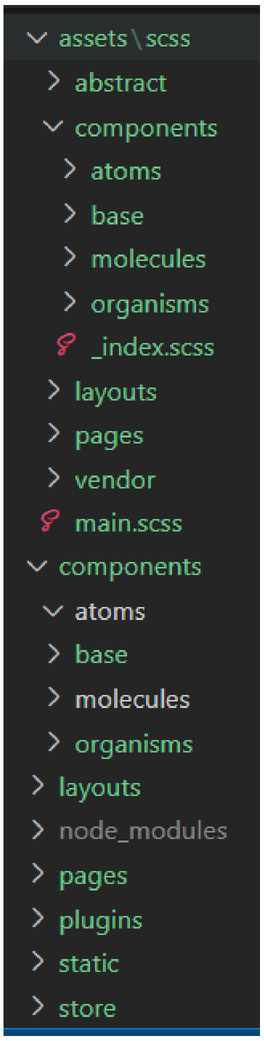


Fig. 3: Draft of design of Form-Builder

7.3 Development

7.3.1 Setting up the workspace

Once the design was ready, the development work started. Like every other development project, it started with setting up the technology stack being used. First, installation of Nuxt, on which this project was primarily based, then incorporating SaaS which was used to manage CSS and installing plugins. One of the major reason why Nuxt was chosen for this project is, besides its server side rendering feature, the routing in Nuxt was very easy, no configuration was required, no extra line of codes were required to establish the routes, each file in pages directory was automatically established as a route, and it came with a standard directory structure which makes developer's job uncomplicated and the keeps the code-base easily readable and clean. Managements of the folders followed atomic-structure, a concept of dividing the files into atoms, molecules and organisms depending on their role, for both SaaS and Nuxt which helps in code management especially when the project is of production level. The following figure (Fig.4) shows the directory structure and as we can see SaaS folder has sub-folders, for example abstract had CSS related to typology, colors, base-margin and base-height used in the platform which allowed the use of variables and kept the CSS clean and managed. Similarly, it had different folder corresponding to each Vue folders to keep their corresponding CSS.



```
assets\scss
  abstract
  components
    atoms
    base
    molecules
    organisms
    _index.scss
  layouts
  pages
  vendor
  main.scss
components
  atoms
  base
  molecules
  organisms
layouts
node_modules
pages
plugins
static
store
```

Fig. 4: Folder Structure

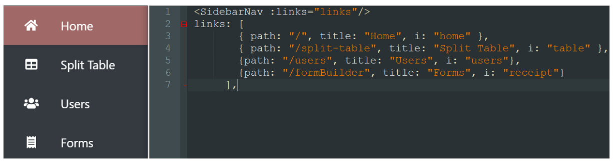
7.3.2 Header and Sidebar

Before going into the development of components, a header and a sidebar was required. Nuxt provides a folder called layouts where different layout for different pages can be stored. For this project, a single layout was used for both form builder page and table page, a layout consisting of header and a sidebar.

```
<nuxt-link /> is used for routing and traverse
through different pages.
```

Header Component and Sidebar Component were created, Sidebar Component having all the links of route passed as props to it. Font-awesome library was used to import icons used in the project, that were also used alongside the links in the sidebar (Fig 5). Sidebar footer consisted of Collapse

Sidebar text which allowed users to hide or show sidebar as they wished for. On sidebar collapse, only icons were shown.



```
<SidebarNav :links="links"/>
links: [
        { path: "/", title: "Home", i: "home" },
        { path: "/split-table", title: "Split Table", i: "table" },
        {path: "/users", title: "Users", i: "users"},
        {path: "/formBuilder", title: "Forms", i: "receipt"}
      ],
```

Fig. 5: Sidebar

7.3.3 Development of Table Component

After having the layout of the project, development of table-Component began since it also meant developing all the base components which were later going to be used for the Form-Builder too. Under the base folder, BaseButton, BaseModal and BaseTable were developed using b-button, b-modal and b-table of bootsrap-Vue framework as a underlying component respectively. For BaseModal, id and title of that modal were passed as a prop [14], default footer was hidden, a binding of computed value was done for it's opening and closing and slots were used to represent the different content that was to be shown depending on where those modals were going to be used for, meaning the slots were occupied by the content passed from a parent component making them scalable and reusable. Similarly, in case of BaseTable data (items and fields) were passed as a prop, item representing content of row and fields representing header of each column. This data was obtained from Vuex store with help of mapGetters as shown below:

```
 ...mapGetters({
      loadItems: "users/getUsers",
      loadFields: "users/getFields",
   }),
```

Before loading the actual data in the table, a skeleton loader was displayed for couple of seconds using b-table's empty slot feature. For actions (edit and delete) and filter table, slots were used so that everything for this base component would be dynamic.

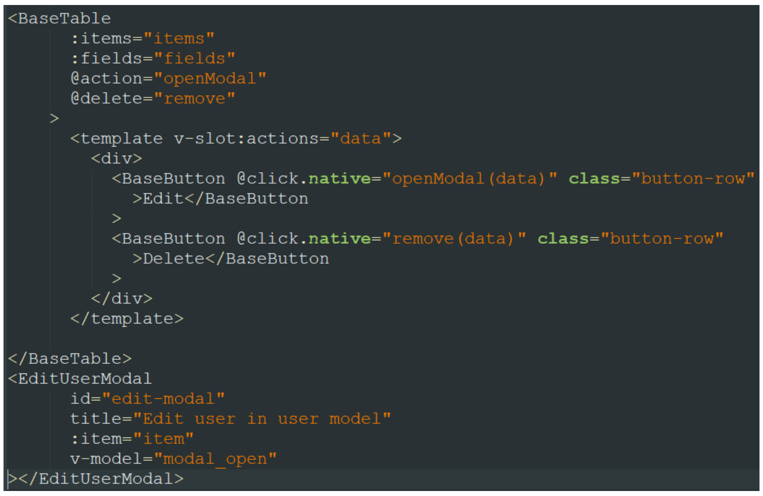
The slots used for actions, skeleton loader and filter table can be shown below respectively:

```
<template v-for="
    slot in Object.keys($scopedSlots)"
    v-slot:[getCell(slot)]="data">
    <slot v-bind="data" :name="slot" />
</template>
```

```
<template #empty>
    <slot name="empty-content">
        <b-skeleton-table
            :rows="5"
            :columns="4"
            :table-props=
            "{ bordered: true,
                striped: true }"
          ></b-skeleton-table>
     </slot>
</template>
```

```
<slot name="filter-content"
    v-if="!showFilters"></slot>
```

Moving into the development of actual table component, in this case UserTable, was developed which used BaseTable as it's child component. The content for slots defined previously were provided from this component. The use of BaseButton was made for every button used here for example for deleting the row of table, resting the filters in filter table, edit and add user button . And on clicking edit and add button, modals were opened that dispalyed forms for users to input value, these modals were respectively called EditUserModal and AddUserModal which were constructed using BaseModal. The snippet of code in UserTable component making use of BaseButton and BaseModal is shown in following figure (fig. 6):



```
<BaseTable
      :items="items"
      :fields="fields"
      @action="openModal"
      @delete="remove"
    >
      <template v-slot:actions="data">
        <div>
          <BaseButton @click.native="openModal(data)" class="button-row"
            >Edit</BaseButton
          >
          <BaseButton @click.native="remove(data)" class="button-row"
            >Delete</BaseButton
          >
        </div>
      </template>

</BaseTable>
<EditUserModal
      id="edit-modal"
      title="Edit user in user model"
      :item="item"
      v-model="modal_open"
></EditUserModal>
```

Fig. 6: Code snippet of UserTable

And finally, the parent component was used as:

```
<UsersTable
    :users="users"
    :fields="fields">
</UsersTable>
```

The deleting of item was handled by Vuex store too, with use of Actions, shown as below:

```
...mapActions({delete: 'users/deleteUser'})
```

Here is the functional structure of tableComponent shown as summary in fig.7:

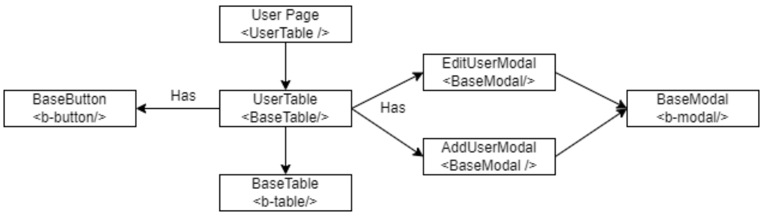


Fig. 7: Functional structure of tableComponent

7.3.4 Development of Form Builder Component

The backbone of this component was, Vue-formulate library, a developer's ease which not only allows to build forms easily but also has built-in validations, styling control and form generator along with other important features. The basic fields that were used in forms, for example, text-input, radio button, select, Date, Measurement, Paragraph etc, were built on right side of the page also called as drag-zone and the other side was built as a drop-area. Vue draggable [15] library was used to make each of those fields drag-gable. The drop area was where the form element were established and it can be represented by following code snippet:

```
<draggable
          class="list-group dropCol "
          :list="forms"
          :group="{ name: 'myGroup' }"
        >
```

The form was schema-based, meaning each of these fields were a separate schema. The list passed as a prop was the list of all the schemas that represented each field situated in drag zone. The drag zone can be shown in the code snippet below:

```
 <draggable
          class="list-group"
          :list="formSchemas"
          :group="{ name: 'myGroup', pull
          :clone="clone"
          @end="preview"
        >
```

Once the fields were dragged, a cloning [16] property was used, meaning the component could be duplicated with unique id, so that same field could be dragged as many times as the user want and the forms could have as many same item. It can be seen in the snippet above with the use of clone passed as a prop to draggable component.

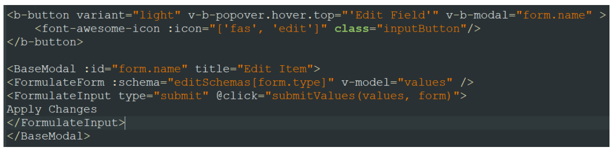
The schemas were imported from store, with use of getters established in that store:

```
 ...mapGetters({
      formSchema:
      "schemas/getFormschemas",
    }),
```

Each of these field draged into drop zone had required property enabled with switch button, so that users could classify which one of those field couldn't be left empty while filling the form. And this was done using ¡b-form-checkbox/¿. This element was bound with a boolean property which was placed under watch, so that every time its true, that specific field was made obligatory. A delete icon was also added to each item, allowing users to modify their forms as per their requirements. The font-awesome icon was wrapped around a button, which when clicked called a function, its parameter being the selected form object, which removed that object from a list of form schemas.

```
 ...mapGetters({
      formSchema:
      "schemas/getFormschemas",
    }),
```

Moreover, each fields were associated with edit button that allowed users to edit features of those fields like label or placeholder. For this to happen, on clicking the button, modal containing a edit form schema was opened and the inputs by users were recorded. For the modal BaseModal and for the form FormulateForm was used which can be shown in the following figure (Fig. 8):

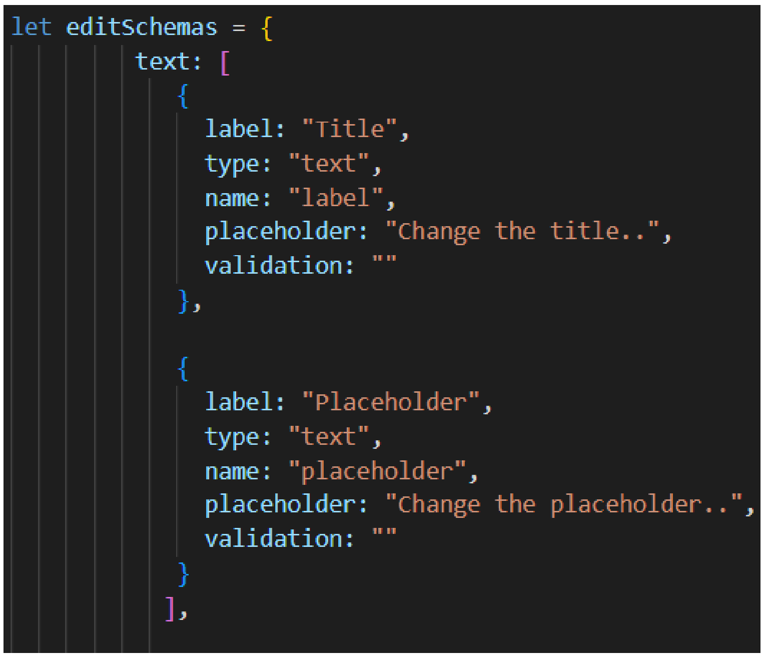


```
<b-button variant="light" v-b-popover.hover.top="'Edit Field'" v-b-modal="form.name" >
    <font-awesome-icon :icon="['fas', 'edit']" class="inputButton"/>
</b-button>

<BaseModal :id="form.name" title="Edit Item">
<FormulateForm :schema="editSchemas[form.type]" v-model="values" />
<FormulateInput type="submit" @click="submitValues(values, form)">
Apply Changes
</FormulateInput>
</BaseModal>
```

Fig. 8: Edit of form element

And this is the schema passed to FormulateForm to be able to edit (Fig. 9)



```
let editSchemas = {
        text: [
          {
            label: "Title",
            type: "text",
            name: "label",
            placeholder: "Change the title..",
            validation: ""
          },

          {
            label: "Placeholder",
            type: "text",
            name: "placeholder",
            placeholder: "Change the placeholder..",
            validation: ""
          }
         ],
```

Fig. 9: Schema for editing form object

In order for user to see how the final created form would look, a Preview button was created using BaseButton, which on emitting click event would open a modal, again built using BaseModal, displaying how the form would look. This displayed form was created with FormulateForm shown as below:

```
<FormulateForm
    :schema="[form]"
    v-model="values"\>
```

As it can be seen that form has a two-way binding property called values. Values is an object which stores all the inputs of the form which later is used in viewing submitted form. This led to developing View Form button again following the same mechanism and the values stored in previously mentioned property are shown when clicking this button which was again based on BaseButton and BaseModal.

One of the most important parts of building forms is validation associated with it. Thanks to the Vue-formulate library, it came built in with it, so it was not necessary to establish our own validations, the only line of code needed to enable validation was having validation property in the schema for each field, which was set to required when users clicked the required switch mentioned above. On the required field being true, the validations were set automatically depending on the input type of the schema.

Finally, all the CSS used for developing these two components were stored in SCSS folder under assets and were categorised depending on the Vue files they were associated with.

8 Results

As a result of completion of project, a form builder component and table component was developed with Nuxt and SaaS as a major technology stack. The project follows atomic-structure as mentioned during planification phase, along with the usage of reusable Nuxt components. Users can now build their own forms as per their requirement with the help of drag and drop feature developed in the project. They have an option to have their own desired fields with the choice of being able to edit those fields (for example being able to change the labels of the field), making those fields to be obligatory or optional and being able to arrange them however they want. Before submitting the form values, they also get to see the Preview of their design and after submitting the submitted form.

The figure in Appendix B shows the end result of form component (Fig. 12).

The preview of the form looks like in the figure shown in Appendix B (Fig.13) and the saved form looks like as shown in Appendix B (Fig.14)

Regarding the table Component, it had features like adding new item to the table, editing and deleting existing item. Moreover, users could filter the table with all the column labels it had, as well sort each column. This can be illustrated in the Appendix B (Fig. 15):

The add (Fig.16) and edit (Fig. 17) feature works as shown in the Appendix B:

9 Conclusion

Nuxt has proven to be a great substitute for frameworks like Vue, React [17], Angular [18] and so on. It is component based and is a very easy tool for developers to use to manage the production level projects, thanks to its standard folder structure. As a part of discovering its usage, accessibility and performance, a form builder and table components were developed. Different dependencies and libraries were used, not to mention the incorporation with SaaS which proved that Nuxt was easily accessible. The built component had user friendly interface that allowed users to build their own forms and access the table easily.

Initially this project was carried out to meet the specific requirements of a company. The form builder was needed in the research platform of company to capture the data stored by it's client obtained during the clinical trials performed on different medical cases and table component was developed to replace all the tables of company's platform by the customised table, meaning no more directly using the table of bootstrap framework. Due to change in the plans midway, even though the correspondence with the company changed, since the project was almost at the end, it was aimed at finishing it. This led to change in the objectives and the duration of project development previously declared, for example the design was no more done or approved by the designers, testing phase was shortened, deployment phase was eliminated and development phase was lengthen.

However, this project has been about understanding the compatibility and accessibility of Nuxt and it would be safe to say that it has been acquired through the development of Form-builder and Table component. These components can be used across various platform and are reusable as they have been based on the custom made base components. The atomic structure had been followed as planned providing easy development path for developer. And a user friendly interface had been achieved to provide better UX.

10 Acknowledgment

I would like to earnestly acknowledge the sincere efforts and valuable time given by my tutor Sara Estravis Nieto, Department of Information and Communications Engineering. The valuable guidance and feedback I received during the period helped me in completing this project successfully.

References

[1] Nuxt, "Nuxt Documentation", Apr 9,2022. Accessed on: Feb 23, 2022. Available: https://nuxtjs.org/docs/get-started/installation

[2] SaaS, "SaaS Documentation", Apr 8,2022. Accessed on: Feb 23, 2022. Available: https://sass-lang.com/guide

[3] Bert Boss, "Cascading Style Sheet", Apr 6, 2022. Accessed on: Apr 8, 2022. Available: https://www.w3.org/Style/CSS/Overview.en.html

[4] bootstrapVue, "bootstrapVue Documentation". Accessed on: Feb 23, 2022. Available: https://bootstrap-Vue.org/docs

[5] bootstrap, "bootstrap Documentation". Accessed on: Feb 23, 2022. Available: https://bootstrap.org/docs

[6] Vue Formulate, "Vue formulate documentation". Accessed on: Apr 4, 2022. Available: https://Vueformulate.com

[7] Scrum, "Scrum", Accessed on: Feb 26, 2022. Available: https://www.mountaingoatsoftware.com/agile/scrum

[8] Vue, "Vue guide", Acessed on: Apr 1, 2022. Available: https://Vuejs.org/guide/introduction.html

[9] SEO, "Search Engine LAnd", Accessed on: Apr 8, 2022. Available: https://searchengineland.com/guide/what-is-seo

[10] Vuex, "Vuex documentation". Accessed on: Apr 6, 2022. Available: https://Vuex.Vuejs.org/guide/

[11] Modal, ”Bootstrap Documentation”. Accessed on: Apr 6, 2022. Available: https://bootstrap-vue.org/docs/components/modal

[12] Bagus Fikri, Fikri Studio. Accessed on: feb 24, 2022. Available: https://dribbble.com/shots/16125238-Form-Builder-Tiimi-HRM

[13] Schema, ”Schema-based Form Generator For Vue.js”. Accessed on: Apr 4, 2022. Available: https://www.Vuescript.com/schema-based-form-generator-Vue-js/

[14] Props, ”Vue documentation”. Accessed on: Apr 6, 2022. https://vuejs.org/guide/components/props.html

[15] SortableJs, ”Sortablejs/Vuedraggable”, May 14, 2021. Accessed on: Apr 6, 2022. Available: https://github.com/SortableJS/Vue.Draggable

[16] SortableJs, ”Sortablejs/Vuedraggable”, May 14, 2021. Accessed on: Apr 6, 2022. Available: https://sortablejs.github.io/Vue.draggable.next/custom-clone

[17] React, ”React Documentation”. Accessed on: May 16, 2022. Available: https://reactjs.org/docs/getting-started.html

[18] Angular, ”Angular Documentation”. Accessed on: May 16, 2022. Available: https://angular.io/start

11 Appendix A

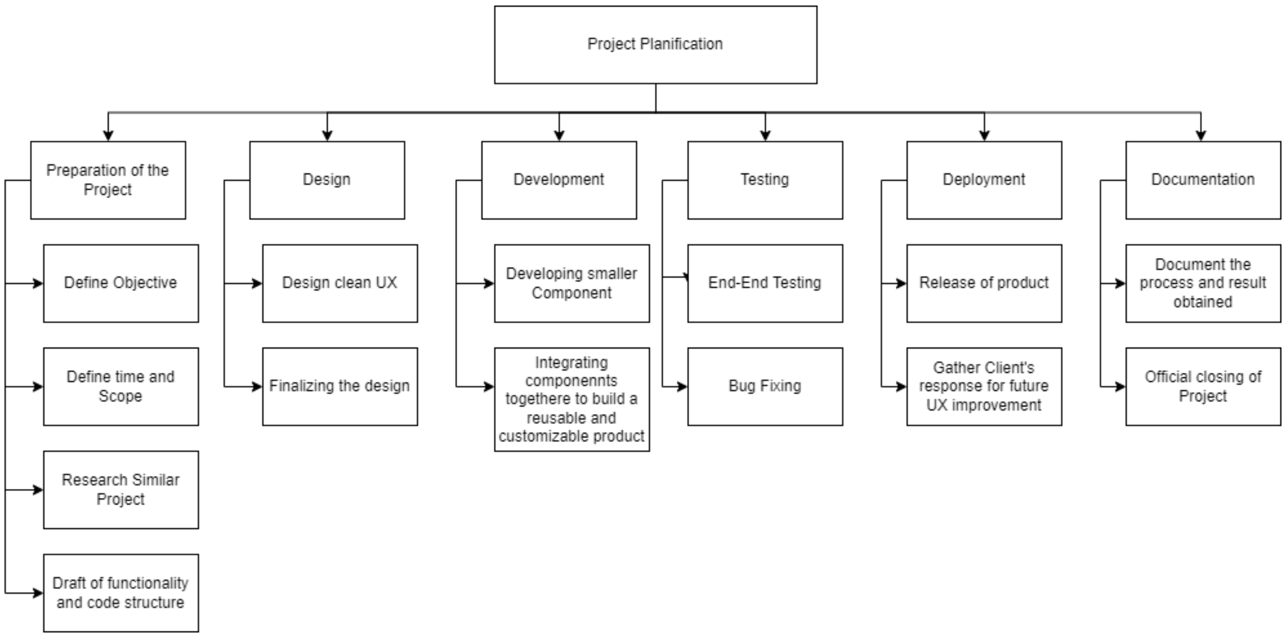
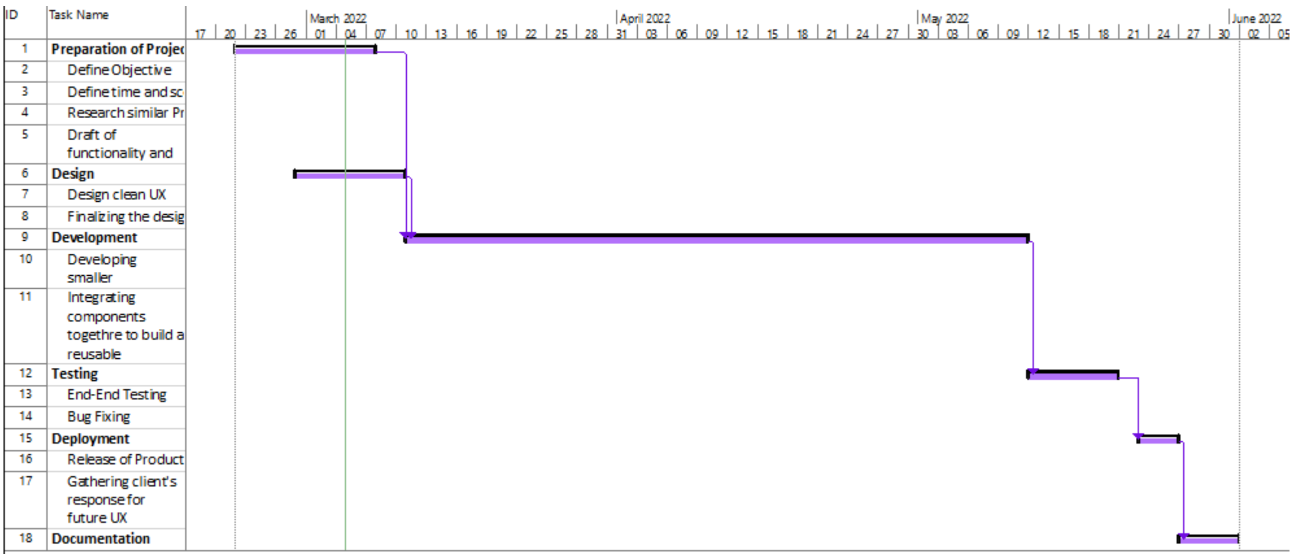

Project Planification
Preparation of the Project
Define Objective
Define time and Scope
Research Similar Project
Draft of functionality and code structure
Design
Design clean UX
Finalizing the design
Development
Developing smaller Component
Integrating componennts togethere to build a reusable and customizable product
Testing
End-End Testing
Bug Fixing
Deployment
Release of product
Gather Client's response for future UX improvement
Documentation
Document the process and result obtained
Official closing of Project

Fig. 10: Work Breakdown Structure



ID	Task Name
1	**Preparation of Projec**
2	Define Objective
3	Define time and sc
4	Research similar Pr
5	Draft of functionality and
6	**Design**
7	Design clean UX
8	Finalizing the desig
9	**Development**
10	Developing smaller
11	Integrating components togethre to build a reusable
12	**Testing**
13	End-End Testing
14	Bug Fixing
15	**Deployment**
16	Release of Product
17	Gathering client's response for future UX
18	**Documentation**

Fig. 11: Gantt Diagram

12 Appendix B

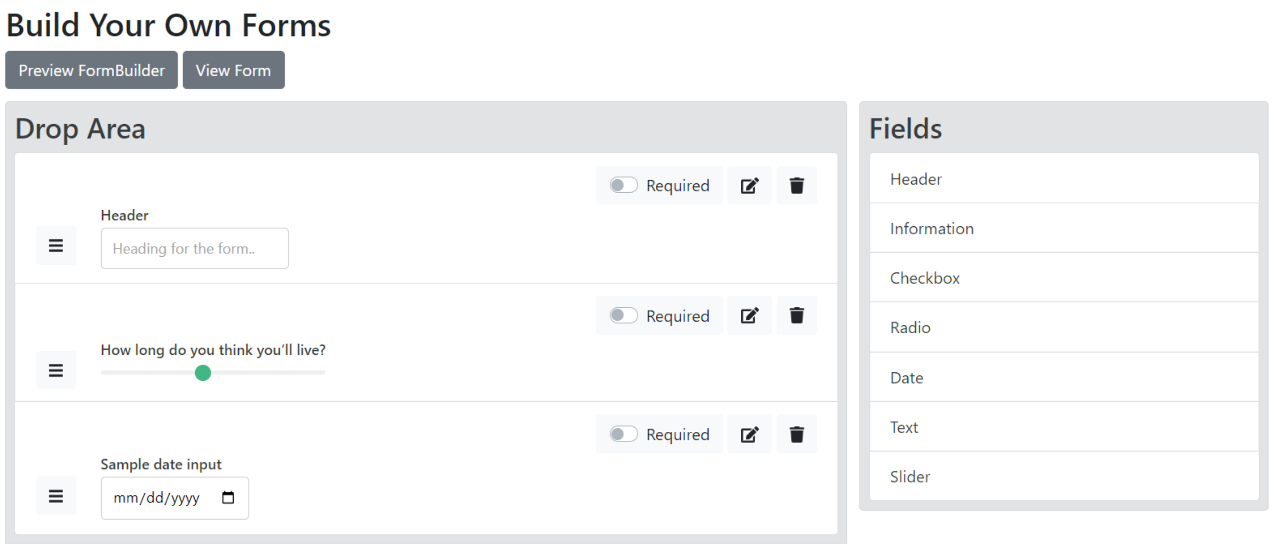


Fig. 12: Form Builder

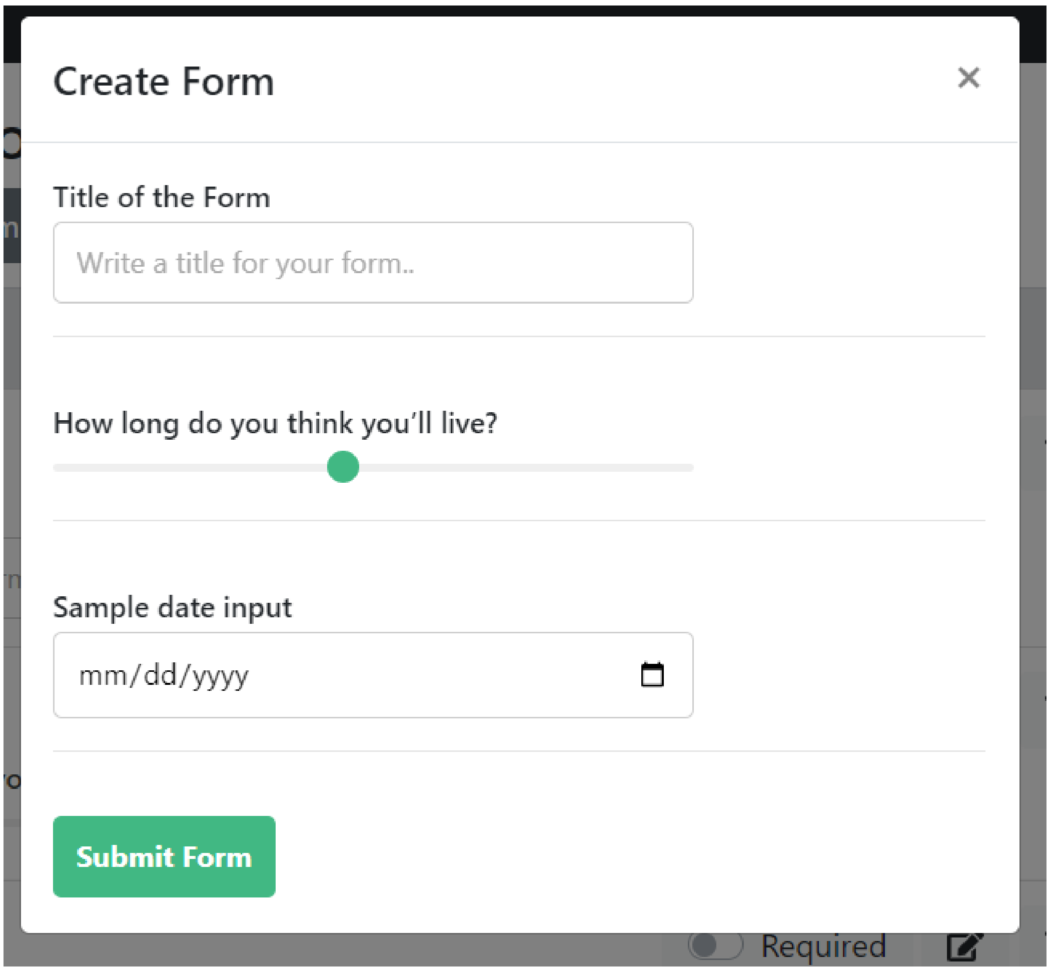
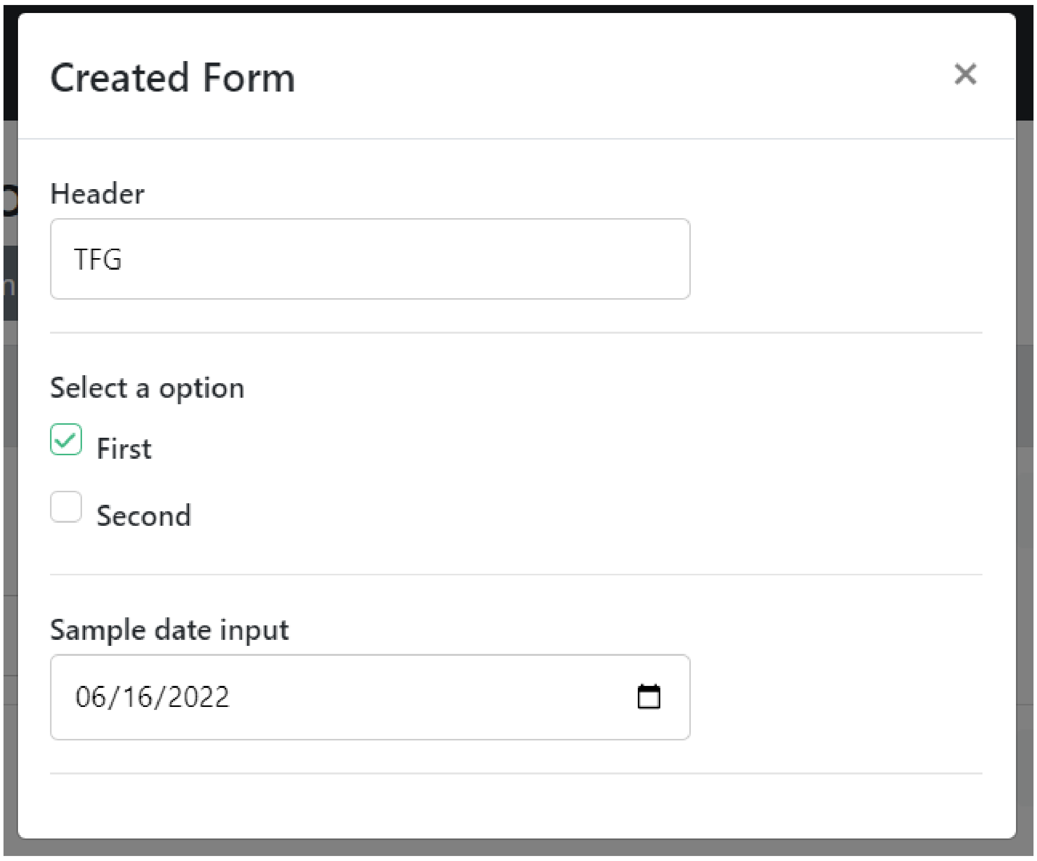


Fig. 13: Preview Form



Created Form

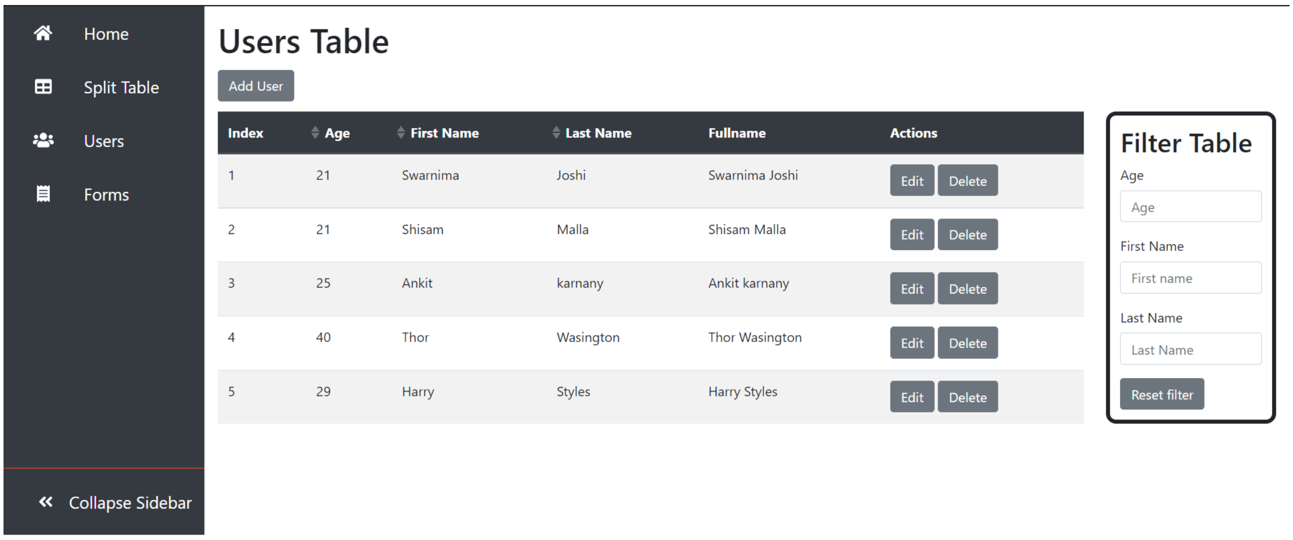
Header

TFG

Select a option
[✓ First]
Second

Sample date input: 06/16/2022

Fig. 14: Saved Form



Users Table

Add User

Index	Age	First Name	Last Name	Fullname	Actions
1	21	Swarnima	Joshi	Swarnima Joshi	Edit Delete
2	21	Shisam	Malla	Shisam Malla	Edit Delete
3	25	Ankit	karnany	Ankit karnany	Edit Delete
4	40	Thor	Wasington	Thor Wasington	Edit Delete
5	29	Harry	Styles	Harry Styles	Edit Delete

Filter Table

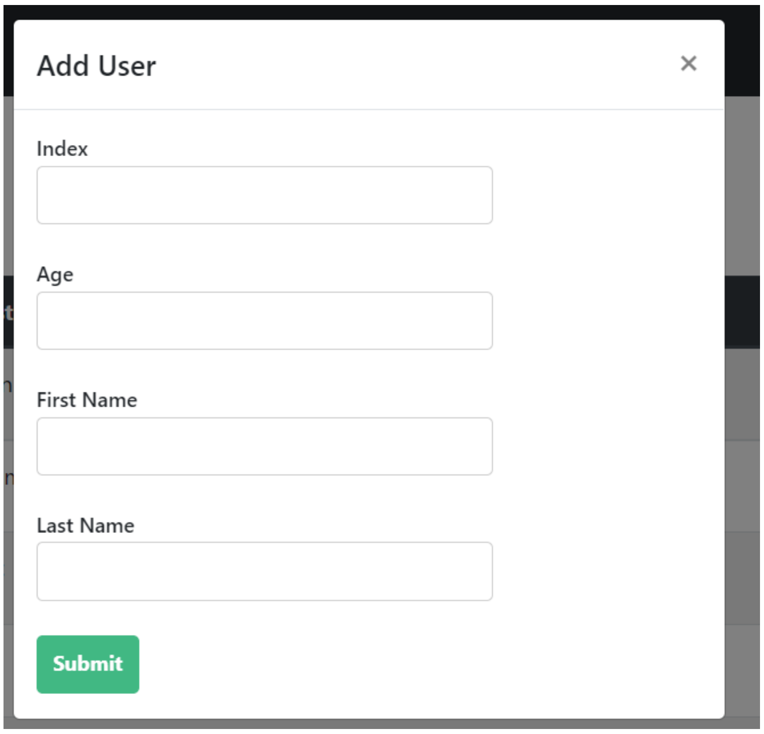
Age: Age

First Name: First name

Last Name: Last Name

Reset filter

Fig. 15: User Table


Add User

Index

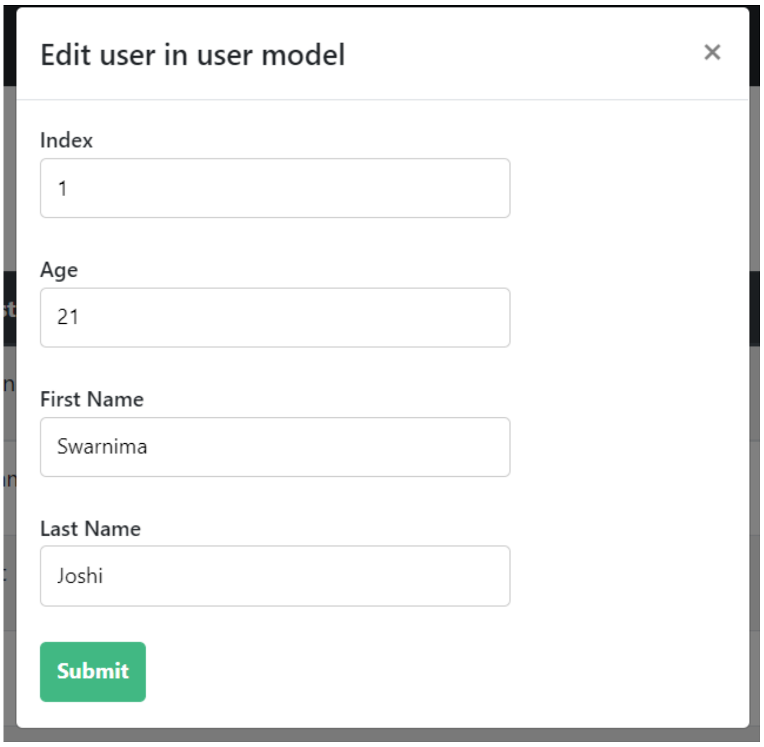
Age

First Name

Last Name

Submit

Fig. 16: Add User Feature


Edit user in user model

Index: 1

Age: 21

First Name: Swarnima

Last Name: Joshi

Submit

Fig. 17: Edit User Feature