Vue model: We use v-model=”some\_variable” to take input from fields that are text fields, text area/multiline text fields, date fields, and numerical fields (donation goal and amount donated by a user during their donation. Additionally, some variable tends to be of the form data.variable\_name.

Data object: we create a data object that is of the type of page we want to store. The data object contains all of the useful fields describing that page including its id which is generated at random and is thus referred as a cuid. The type object of the data object describes the data types of each attribute of the data object. For example for the type Page, we have

type Page = {

cuid: string,

page\_name: string,

day\_of\_birth: Date,

donation\_goal: Number,

…

};

The page is later initialized to

const data = ref<Page>({

cuid: “”,

page\_name: “”,

day\_of\_birth: new Date(),

donation\_goal: 0,

…

});

Next, we use an asyncronous function containing a useFetch function to call the backend.

const getDataPage = async(id:string) => {

    const {data : pageDataDB} = await useFetch('/api/page', {

    method: 'GET',

    query: {cuid:id}

})

// Method that saves form data to the database for a page that has cuid: router.params.id

const save = async () => {

await useFetch('/api/page', {

// Checks if there is a prexisting page to edit or if to create a new page

method: router.params.id !== "0" ? 'PUT' : 'POST',

body: ({ ...data.value, family\_cuid: family\_cuid\_data, cuid: router.params.id as string })

}

)

};

.

In the backend which is one of the api handlers, we grab the data from the front end from an awaited readBody() on an even for put, post, and delete requests. For get requests, we use an awaited getQuery() an event with an object destructure on the variables we are forwarding from frontend to backend. So an example of readBody usage is const body = await readBody(event). To grab an attribute from the body object we use body.variable\_name. On the other hand, for get requests, we say const { cuid } = getQuery(event) .

Finally, we call a prisma statement to call the database. Make sure to use the type assertion as data\_type in order to comply with Typescript when filing out database calls for using a value from a ref or computed ref. The basic format of prisma’s queries is to make an awaited prisma.table/model\_name.method\_name. The methods are create() for creating new entries in the database or post requests, update() for changing an entry in the database, findFirst() for finding a single entry, findMany() for finding multiple entries of a table such as all the pages that belong to a family or advocate, delete() for removing an entry in the database, and deleteMany() for deleting multiple entries in the database.

// updates a pre-existing page

    const queryRes = await prisma.page.update({

      where: {

        cuid: body.cuid

      },

      data: {

        ...body

      }

    });

During the 2023 Summer work period we used watchers for image handling and donation management. A watcher allows the program to wait for a ref to change before executing code in its function body.

watch( previewCuid, async() => {

// code that handles a change in preview image

...

})

We make the 3rd argument { deep: true } if the ref has multiple properties.