# Setting up in local environment

1. Install nodejs [<https://nodejs.org/en/>]
2. set the script to run in development mode
   1. Set **is\_dev = true** at line 4 of ./backend/constants.js
3. [optional] turn on email verification
   1. Set enable\_email\_verification = true at line 5 of ./backend/constants.js
4. Install frontend package and run frontend script
   1. Navigate to directory 🡪 *cd frontend*
   2. Install package 🡪 *npm install*
   3. Run the script 🡪 *npm start*
5. Install backend package and run backend script
   1. Navigate to directory 🡪 *cd backend*
   2. Install package 🡪 *npm install*
   3. Run the script 🡪 *npm start*

# Setting up in NIE server

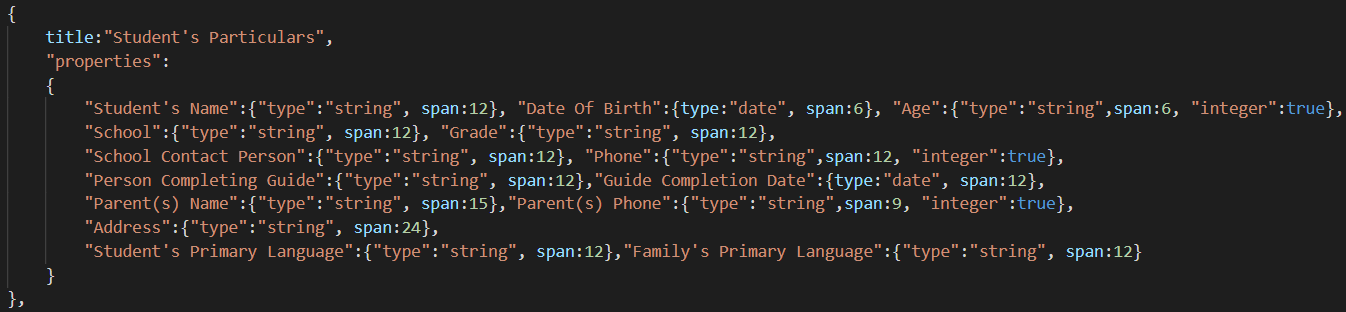
1. In your local environment, do the following
   1. Set **is\_dev = false** at line 4 of ./backend/constants.js
   2. Navigate to backend 🡪 *cd backend*
   3. Remove build folder 🡪 *rm -R build*
   4. Navigate to frontend 🡪 *cd ../frontend*
   5. Build the script 🡪 *npm run-script build*
   6. Move the build folder to backend 🡪 *mv build ../backend*
2. Push the source code to repo
3. Go to <https://console2.rdc.nie.edu.sg/auth/login>
4. Login
5. Go to the ssh terminal
6. Run the following
   1. Navigate to repo directory -> *cd /usr/share/nginx/html/Digitization-of-WATI-forms*
   2. Pull repo 🡪 sudo git pull <https://github.com/pngqj/Digitization-of-WATI-forms>
   3. enter sudo password
   4. Navigate to -> cd backend
   5. Remove the current process from pm2 🡪 pm2 delete app
   6. Start the process 🡪 pm2 add app.js
   7. Check if there is any error (status should be online)🡪 pm2 list
7. Go to <https://atconsideration.rdc.nie.edu.sg/home> to check if your new website is working

# Current Progress of website

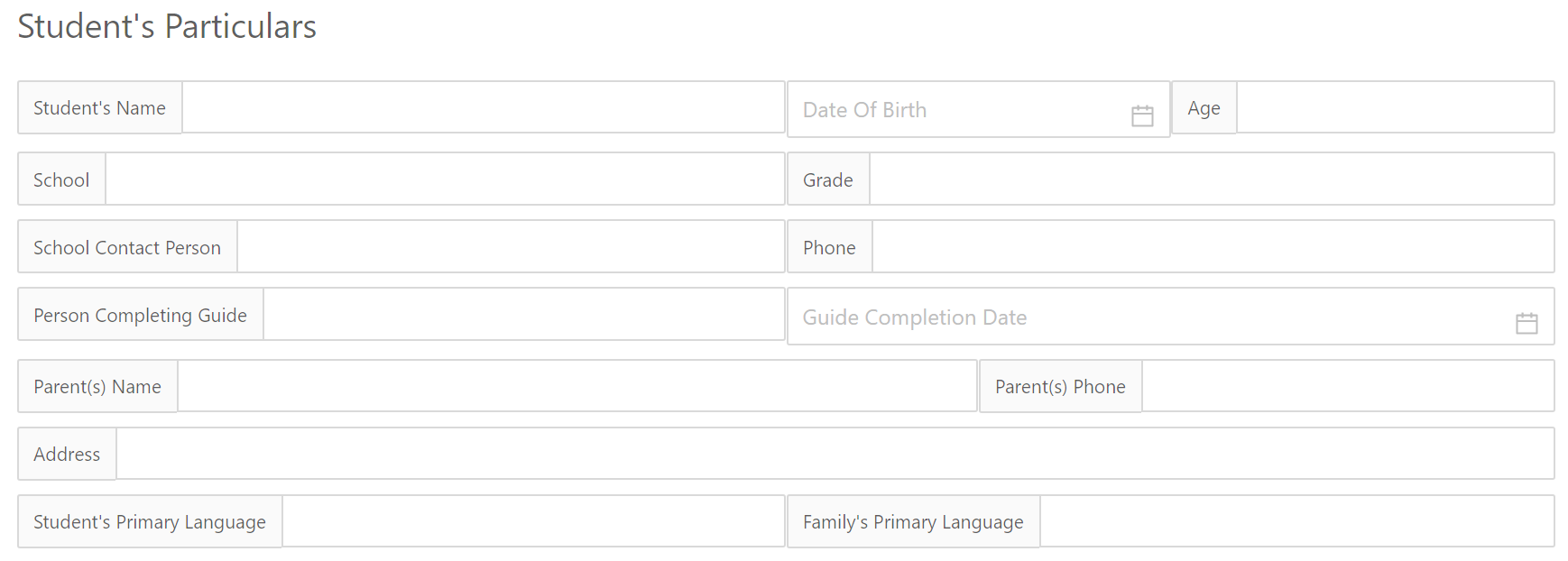
|  |  |  |
| --- | --- | --- |
| Feature | Current Progress | Known Bugs |
| * Homepage with banner and scrolling animations | Done | None |
| * Login | Done | None |
| * Sign up with email verification | Still waiting for Chee Kiong to fix server’s bug  Email verification is temporary turned off. You can turn it on by setting enable\_email\_verification = true at line 5 of ./backend/constants.js | Email verification only works on NIE/NTU email addresses. It will not work on other email domains like Hotmail or Gmail  This is a server issue. I have already contacted Chee Kiong to help with this issue. I am still waiting for his reply  His email address: cheekiong.kuek@nie.edu.sg |
| * Adding a new student * Editing information of student * Deleting student * Sharing student’s information with another user/ teacher | Done | None |
| * Viewing student’s forms * Adding a new form for a student * Editing fields in a form * Deleting a form * Downloading form as PDF | Done | None |
| * Uploading image for a student | This feature was added recently (19 July 2020), so it is still buggy  Temporary removed because it is buggy. Uncomment line 433 of TabManager.js to view it | * Works in localhost, but not working in server * May contain other undiscovered bugs because this feature is not well tested yet |
| * Mobile version of website | Not implemented yet. The skeleton code of this mobile version is available in the source code. Please continue to work on this | NIL |
| * Form customization | Not implemented yet. Ask Dr Wong and Yingmin for more information about this feature. This feature should be possible with the current implementation of the website |  |

# How the form works?

Frontend will generate a form based on JSON Schema. For example, the JSON schema in **Fig 1** will generate the form in **Fig 2**



**Fig 1**



**Fig 2**

Check out the live playground in the link below. The concept used in this GitHub is quite similar to the WATI system I have developed.

<https://rjsf-team.github.io/react-jsonschema-form/>

The system will save all data inputted into the form by end-users in JSON format

**Why JSON schema instead of standard HTML fields with SQL?**

A few reasons

1. There are 10 type of form. If I use SQL to store the form data, I will need to create 10 tables to manage all the forms. It’s easier to use NoSQL or JSON to manage this.
2. Dr Wong mentioned that we may need to develop a feature that allows end-users to create custom forms (ask him for more info). I think it’s easier to develop this feature with NoSQL or JSON
3. Everything can be written in JavaScript! No need for additional language like SQL

# JSON Schema Explanation

**Forms and Sections**

[

{

//section 1

“title”: “section 1 title”,

“properties”: {

/\*section 1 properties\*/

“property1” : {“type”: “string”},

“property2” : {“type”: “date”}

}

},

{

//section 2

“title”: “section 2 title”,

“properties”: {

/\*section 2 properties\*/

“property1” : {“type”: “string”},

“property2” : {“type”: “boolean”}

}

}

]

Each form is represented as an JSON array of JSON objects, as shown above. One JSON object represents one section of a form. For example, Fig 1 represents the “Student's Particulars” section of the “Referral/Question Identification Guide” form.

Each section contains a title and properties of the section.

**Section Properties**

“properties” contains the elements of the forms (e.g. Checkbox, Table, Input field, etc).

* The key of each element is the name of the element.
* The value of each element is a JSON object. It contains the attributes of the element. The table below shows the different attributes of said object

|  |  |  |
| --- | --- | --- |
| Attribute | Purpose | Possible values |
| type | The type of element (e.g. Checkbox, Table, Input). | * “string” * “date” * “boolean” * “boolean string” * “boolean section” * “table” * “paragraph” * “select” * “switch” |
| span | An integer to represent the width of the element. Range from 1 to 24. For example, an element that occupies 33% of the width will have a span of 8 | * integer value from 1 to 24 |
| For “string” type elements | | |
| integer | If true, users can only input a number into the element. Cannot be used with “long”. | * true or false |
| long | If true, users can type large amount of text into the element. Cannot be used with “integer”. | * true or false |
| highlight | Highlight the string element. Can be used with “long” or “integer” | * true or false |
| For “paragraph” type element | | |
| description | The text to be displayed on the “paragraph” element | * A string |
| isHTML | If true, read the description as html and display accordingly | * true or false |
| For “table” type elements | | |
| columns | An JSON array that contains the properties of the table.  The implementation is similar to Antd’s table. Click [this link](https://ant.design/components/table/#How-To-Use) to find out more. | A JSON array containing objects with the following attributes:   * “title” * “dataindex” * “editable” * “width” * “colheader” * “inputType” * “isHTML” |
| needCheckBox | If true, display checkbox on the left of each row of the table | * true or false |
| needAddButton | If true, display “add row” button below the table | * true or false |
| For “Boolean section” type elements | | |
| section | The section number of the element. See “Referral Guide.js” for an example | * Any integer >= 1 |
| For “Boolean string” type elements | | |
| isTextArea | If true, allow users to type a large amount of text in the element. | * true or false |
| paragraph | The paragraph of text to be displayed in this element. See “section1.js” for an example | * a string |
| For “select” type elements | | |
| options | The options available for this element. See “section1.js” for an example | * An array containing strings for each option |
| For “switch” type elements | | |
| properties | See “section1.js” for an example | * An object containing the elements to be toggled by the switch |