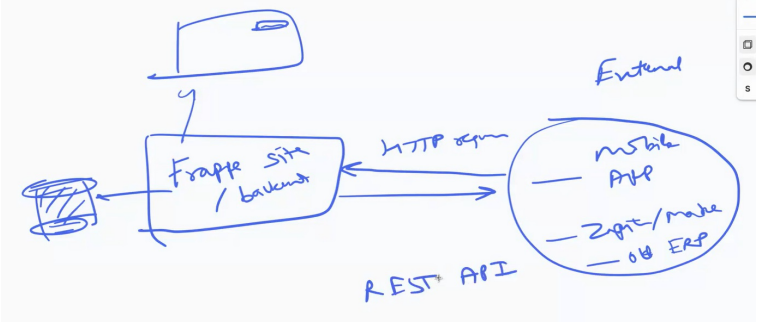
Day 2

Created	@January 12, 2024 3:15 PM
Status	Open
Updated	@September 17, 2024 5:47 PM

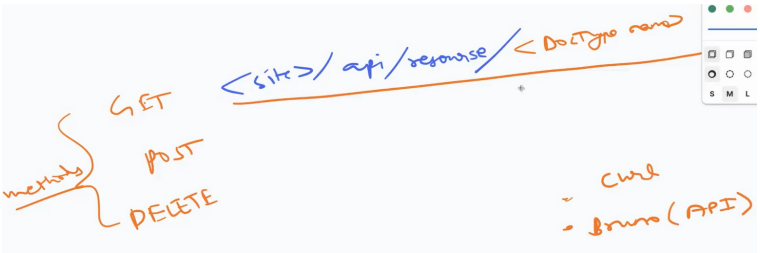
rename_doc → method to rename a document

3 ways to do CRUD → through Desk, REST API (frappe provides out of the box REST API for every doctype we create), or script.

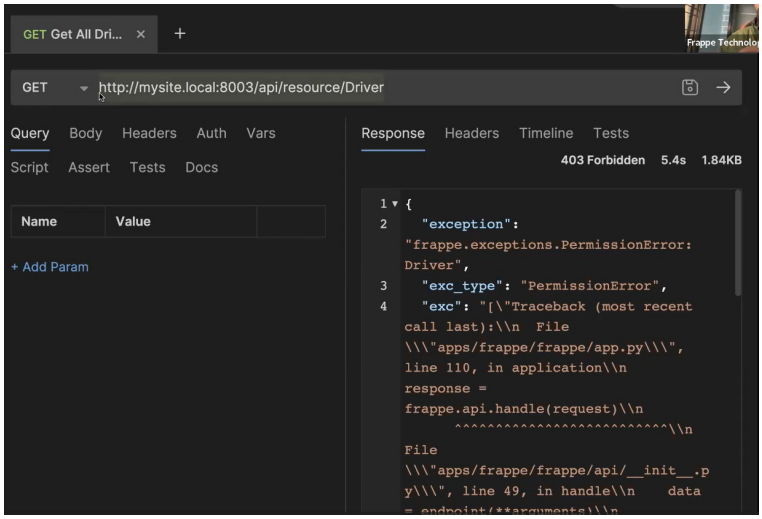
REST APIs:



External systems can talk to our site via HTTP (making HTTP requests), and for this, we define endpoints (REST API).

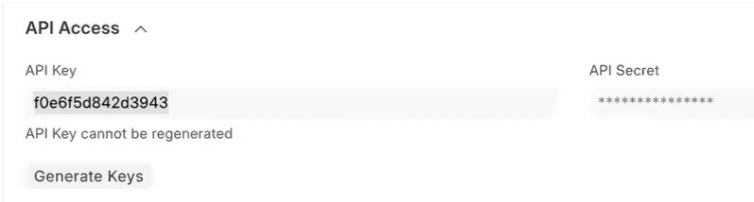


We can make a request through/using the url `<site>/api/resource/<doctype-name>`. Depending on how we make the request, it might create a new data if we want to create a new record, or if we want to just get the list of the doctype records, it will return that.



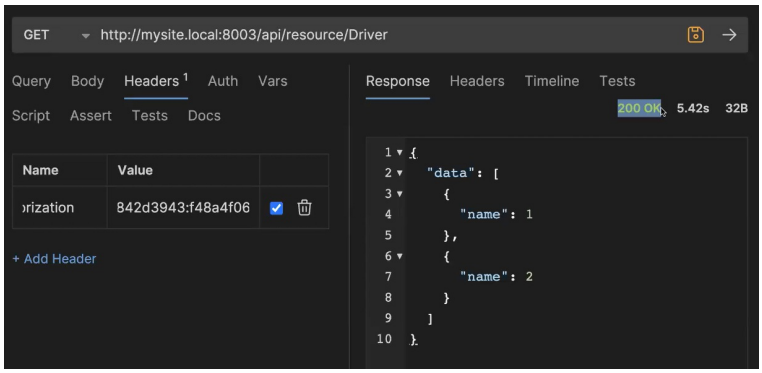
Example: We can make a GET request using the url to get the list of Driver records. However, it will return a permission error and this is what we want because our site data shouldn't be accessible to everyone. So, we need to tell it that we are authorized to make this request.

API Authentication

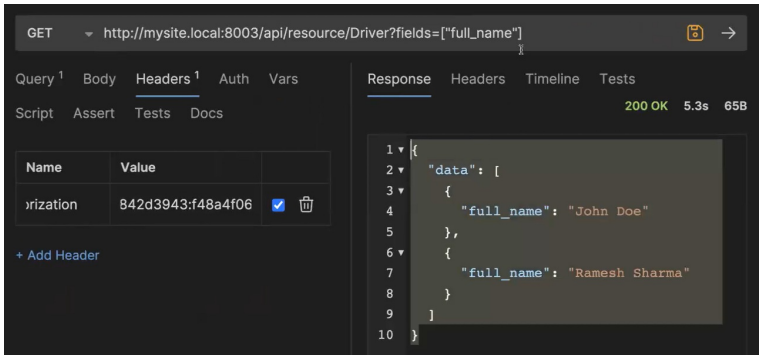


For any and each user, we can get a set of API keys: User → the record of the user → Settings → API Access → Generate Keys. We will get 2 keys (API Secret & API Key), the secret key is only shown once so make sure to save it.

How to authenticate requests: Make a header named `Authorization` (this is a must), then the value must be `token <api-key>:<api-secret>`.

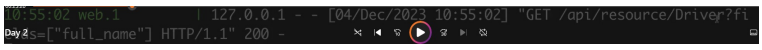


Now the GET request should work, and the default is it will return just the `name` of each records.

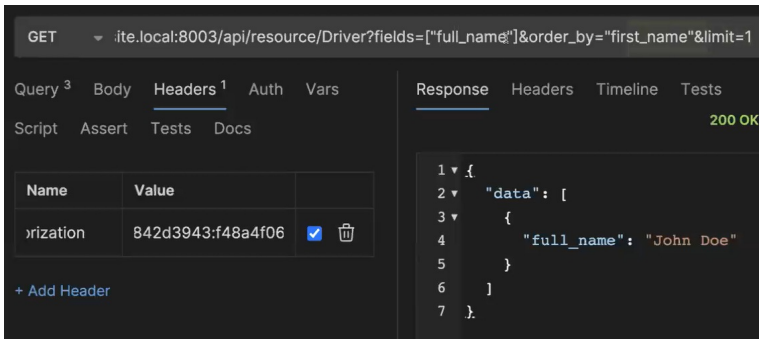


Request Methods Options

We can specify the fields we want by appending `?fields=["field_1", "field_2"]`.

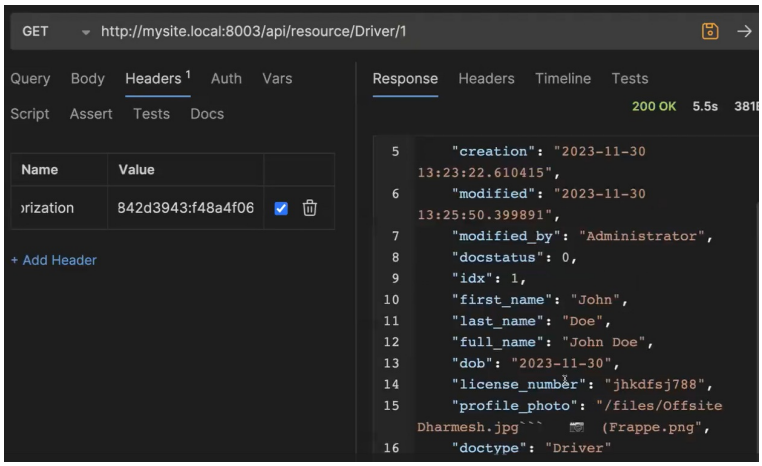


We can also see the GET request for Driver on the output of the bench console, and it shows that the response is 200 (okay).

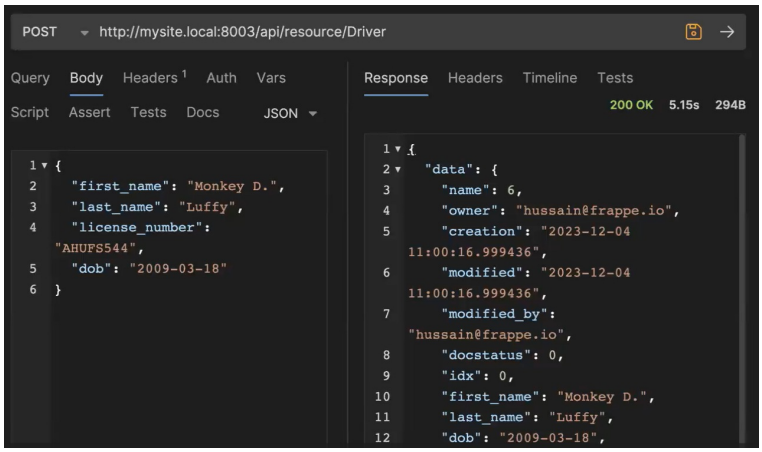


To append other filters, use `&`. Example: `?fields=["field_1", "field_2"]&order_by="field_3"&limit=1`.

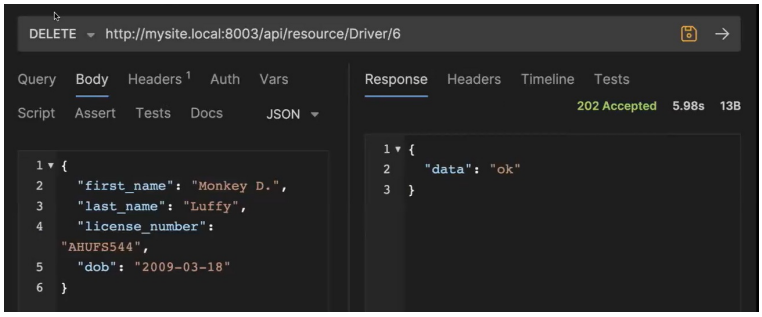
We can also return all the data of a particular record by providing the `name` or `id` of the record. Example:



To create a record (use a POST method to the doctype url), we can provide a `Body` json that provides the key:value pair for all the mandatory fields at a minimum. Example:



We can also delete a record by providing the `name` or `id` of the record. Example:

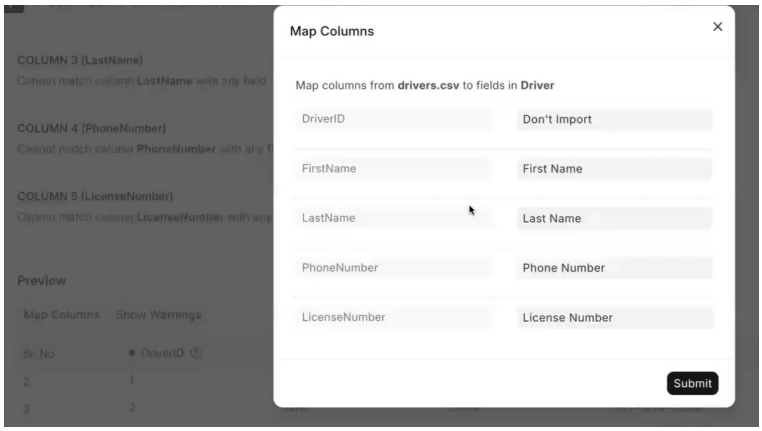


But it is different if we want to upload files (check docs).

We can also use other authentication methods such as OAuth. We can make our site act as an OAuth server where any other site can talk to that site as an OAuth server for login.

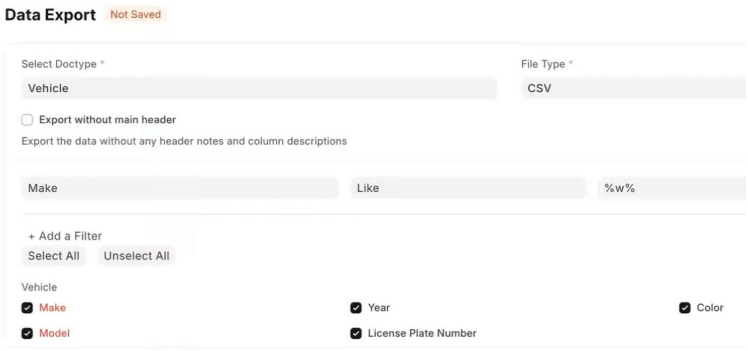
Data Import & Export:

To import data, we have to allow data import on the doctype settings (Form Settings) we want to import. (ctrl + j is a shortcut to jump to any doctype field). Now we can get the doctype on `New Data Import`.

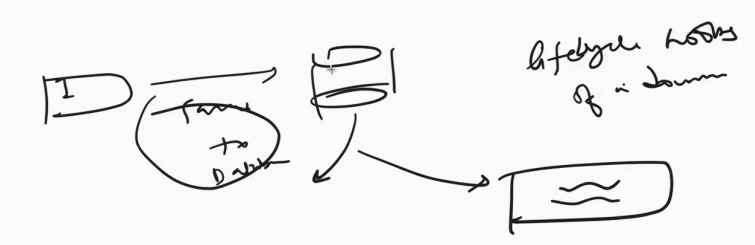


If the column names (on the imported file) does not match the field names of the doctype, we can use `Map Columns` to map them.

Data Import works in the background (make sure to have scheduler enabled).



To export data, we can use `Data Export`. We can even add a filter to the data we are exporting.



Document Hooks:

We can hook into different lifecycles of a document. For example, we can run some code before a record is saved/deleted in the database if we want.

For example, we can programmatically set the full_name of `Driver` before the record is saved on the database:

```
class Driver(Document):
    def before_save(self):
        self.full_name = self.first_name + " " + self.last_name
```

`self` refers to that particular `Driver`, so if the code is running for Driver 1, then self will refer to the Driver 1 object.

`autoname` is used to generate `name` on any script.

Submittable DocType & Link Field:

Amended From field is automatically added for submittable doctype. It's a link field that points to the same doctype.

View Settings → Show Title in Link Fields → it will show the `Title Field` values instead of the `id` or `name` whenever we want to select records in any Link field that points to the doctype

Submittable doctype → There will be a Submit button on every saved records, and if clicked & confirmed, it will change the

docstatus to Submitted (1), and all fields will become `read-only`. If we want to make a field editable even on submit, make sure to check `Allow on Submit` option for it.

There are 3 docstatus: Draft (0) where we can edit fields etc., Submitted (1) the document becomes non-editable, and Cancelled (2) to make the document invalid, it's still uneditable, but we can `Amend` it to create a new editable document that copies all values from the cancelled document + it will be linked to that cancelled document on Amended From field.

Why submittable exists? In general, there are 2 types of doctypes: Master (stuff can change) & Transactional (after some point, it does not make sense that stuff/values can change, e.g Sales Invoice)

View Settings → Search Fields → it will show all fields selected here when we are selecting records on Link field that points to this doctype. It's not only for display, we can search by these fields selected as well.

Child Table/DocType:

To implement one-to-many relationship, we use Child Table/DocType. It will have no list view and can't be searched because they can't exists in isolation (not independent), they need to have a `parent`. However, the table for this doctype will still be created and we can see all the records from this table. Use the field type `Table` that will help **link** the Child DocType to the Parent DocType.

- `parent` : name of the parent.
- `parenttype` : DocType of the parent.
- `parentfield` : Field in the parent that links this child to it.
- `idx` : Sequence (row).

The parent doctype won't have any field that points to the child doctype, but each child doctype records will have multiple special fields that points to the parent record.

On default, the child table on the form view will only show mandatory fields. To show more fields, make sure to check `In List View` for the fields we want to also display.

Python Console & API:

```
In [5]: rb = frappe.get_doc("Ride Booking", "RO-12-2023-0022")

In [6]: rb.name
Out[6]: 'RO-12-2023-0022'
```

```
[<RideBookingItem: 4b1c9dfd65 parent=RO-12-2023-0022>,
 <RideBookingItem: b45219db90 parent=RO-12-2023-0022>]

In [13]: rb.items[0]
Out[13]: <RideBookingItem: 4b1c9dfd65 parent=RO-12-2023-0022>

In [14]: rb.items[0].destination
Out[14]: 'Frappe Office '
```

```
In [3]: frappe.get_all("Ride Booking", fields=["name", "doc
In [3]: frappe.get_all("Ride Booking", fields=["name", "docstatus", "vehicle"], filters={"docstatus": (">", 0)})
Out[3]:
[{'name': 'RO-12-2023-0021',
  'docstatus': 1,
  'vehicle': 'Maserati-Quattroporte-2021-19'},
 {'name': 'RO-12-2023-0020-1',
  'docstatus': 1,
  'vehicle': 'Chevrolet-Impala-2023-14'},
 {'name': 'RO-12-2023-0020',
  'docstatus': 2,
  'vehicle': 'Chevrolet-Impala-2023-14'}]
```

We can play around the Python console that is connected to our Frappe site before we write any logic on the server-side script. We have many APIs such as `frappe.get_doc, frappe.get_all` and `frappe.delete_doc`.

```
In [6]: frappe.get_all("Ride Booking", fields=["name", "docstatus", "vehicle"], filters={"docstatus": (">", 0)}, limit=
   ...: 1, order_by="creation desc")
Out[6]:
[{'name': 'RO-12-2023-0021',
  'docstatus': 1,
  'vehicle': 'Maserati-Quattroporte-2021-19'}]
```

When we play around the console, it does not automatically commit the changes to the database so it's good to play around with. If we want to commit the changes, we can use `frappe.db.commit()`, or combine `.save()` with `frappe.db.commit()`.

```
In [5]: frappe.get_single("Ride App Settings")
Out[5]: <RideAppSettings: Ride App Settings>

In [6]: frappe.get_single("Ride App Settings").price_per_km
Out[6]: 20.0

In [7]: frappe.db.get_single_value("Ride App Settings", "price_per_km")
Out[7]: 20.0
```

Single DocType:

Single DocType does not have a list view, it only have a form view for a **single record**. It does not have its own separate table in the database. Instead, they are all stored in `tabSingles` that stores 3 fields: doctype, field, value.

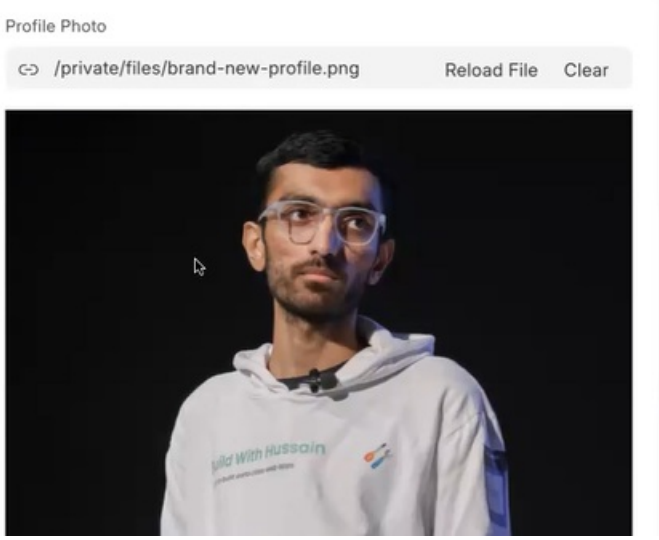
Examples:

Example validation for number of items in child table:

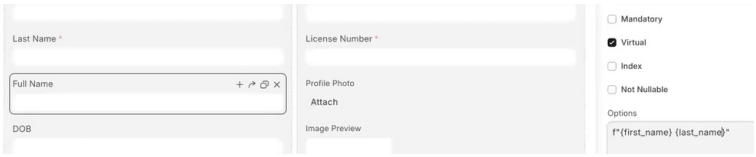
```
if len(self.images) > 5:
    frappe.throw("Only 5 images are allowed")
```

Example for preview image:

Have an image field and refer it (Options) to the attach image fieldname. Result:

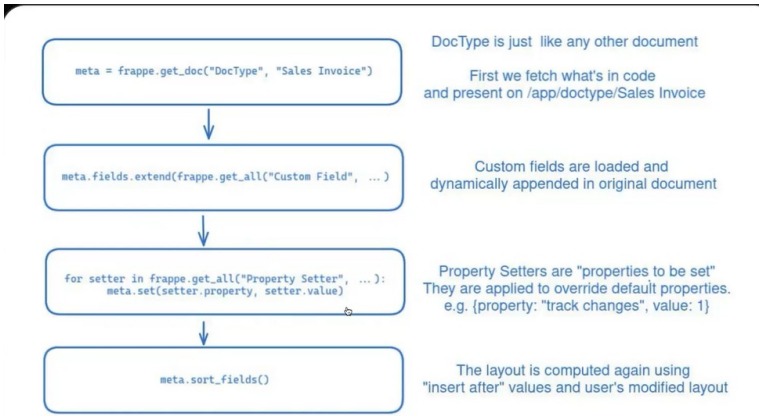


Example use of virtual field:



So, when we check `Virtual` on a field, we can now insert Options to automatically fetch the values of other fields without having to write it on the python script.

In customize form, whatever fields we add will count as Custom Fields, so it won't be updated on the schema/json of the doctype, it will only live on the site.



There are 2 doctypes that tracks customization (on the customize form): `Custom Field` & `Property Setter`.

So, first frappe gets the document as usual, then it fetches the Custom Field for that document. Then, it fetches Property Setter for that document (if any) and overrides the default properties.