Special Workshop in IT

Week 3: DocTypes, Form Scripts, and Portal Pages

Sirasit Lochanachit, PhD

Today's Objectives

1) Creating 3 DocTypes
    a) Library Membership
    b) Library Transaction
    c) Library Settings
2) Customising Form Scripts
3) Creating Portal Pages for Website Visitors

Previously

We have created 2 DocTypes:

1. **Article**: A Book or similar item that can be rented
2. **Library Member**: A user who is subscribed to a membership

- Based on the type of data stored, doctypes can be classified into 2 broad classes: **Master** and **Transactional**
    - Article and Library Member are examples of Master class because they represent an entity (physical or virtual)

Today

We are going to create 3 more DocTypes:

1. **Library Transaction**: Records of an Issue or Return of an article by a Library Member who has an active membership
2. **Library Membership**: A document that represents an active membership of a Library Member
    - Stores transactional data
3. **Library Settings**: Settings that define values like Loan Period and the maximum number of articles that can be issued at a time

- All doctypes are linkable
    - Linked DocTypes are DocTypes that are linked in other doctypes as Link fields.

Librarian Membership DocType

Library Membership DocType

Create a 3rd doctype: **Library Membership**, which have the following fields

- Library Member (Link, Mandatory)
  - Link is similar to a foreign key
  - This field will link to a record of **Library Member** doctype
- Full Name (Data, Read Only)
  - **Fetched from** the 'full_name' field linked to the **Library Member** doctype
- From Date (Date)
- To Date (Date)
- Paid (Check)

It should have **Is Submittable** enabled

Naming set as LMS.##### and this is restricted to **Librarian** role only

In the View settings section, the Title Field should be set to 'full_name'

Submittable Option in a DocType

A doctype with submittable option enabled can have 3 states:

- **Draft**
  - A document or a value of any field can be changed/edited
- **Submitted**
  - Unable to change or edit the document or the value
- **Cancelled**
  - A submitted document can be cancelled

Controller Methods and APIs

Controllers

- Controller methods allow you to write business logic during the lifecycle of a document.
- A Controller is a normal Python class which extends from **frappe.model.Document** base class
  - This base class is the core logic of a DocType.
  - It handles how values are loaded from the database, how they are parsed and saved back to the database.

Controller Methods

When we created a DocType named **Library Member**, a python file named 'library_member.py' is created

- This is where we can add custom methods, using controller hooks

```
def before_save(self):
    self.full_name = f'{self.first_name} {self.last_name or ""}'

def validate(self):
    if self.age > 60:
        frappe.throw('Age must be less than 60')

def after_insert(self):
    frappe.sendmail(recipients=[self.email], message="Thank you for registering!")
```

  - More hooks - https://frappeframework.com/docs/user/en/basics/doctypes/controllers

Controller Hooks

Method Name	Description
before_submit	Called before a document is submitted.
before_cancel	This is called before a submitted document is cancelled.
before_update_after_submit	This is called *before* a submitted document values are updated.
before_insert	This is called before a document is inserted into the database.
before_naming	This is called before the `name` property of the document is set.
autoname	This is an optional method which is called only when it is defined in the controller at document creation. Use this method to customize how the `name` property of the document is set.
validate	Use this method to throw any validation errors and prevent the document from saving.
before_save	This method is called before the document is saved.
after_insert	This is called after the document is inserted into the database.
on_update	This is called when values of an existing document is updated.
on_submit	This is called when a document is submitted.
on_update_after_submit	This is called *when* a submitted document values are updated.
on_cancel	This is called when a submitted is cancelled.
on_change	This is called to indicate that a document's values has been changed.
on_trash	This is called when a document is being deleted.
after_delete	This is called after a document has been deleted.

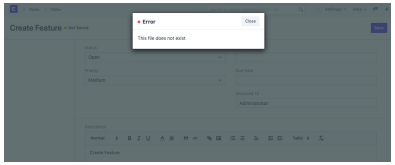
Dialog API (Python) - Shows a message

frappe.msgprint(msg, title, raise_exception, as_table, as_list, indicator, primary_action)

- msg: The message to be displayed
- title: Title of the modal
- as_table: If msg is a list of lists, render as HTML table
- as_list: If msg is a list, render as HTML unordered list
- primary_action: Bind a primary server/client side action
- raise_exception: Exception

Example:

```
frappe.msgprint(
    msg='This file does not exist',
    title='Error',
    raise_exception=FileNotFoundError
)
```



Dialog API (Python) - Throw Error and raise Exception

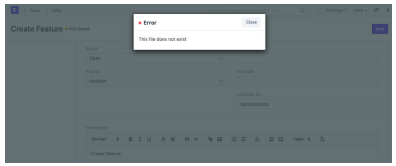
frappe.throw(msg, exc, title)

This method will raise an exception as well as show a message in Desk.

- msg: The message to be displayed
- title: Title of the modal
- exc can be passed an optional exception
  - By default it will raise a ValidationError exception

Example:

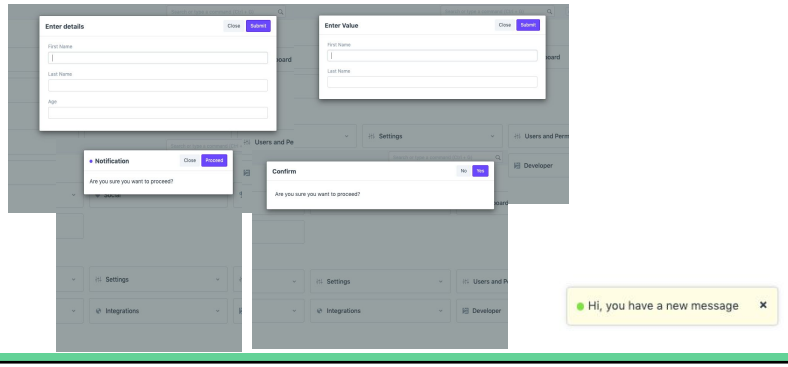
```
frappe.throw(
    title='Error',
    msg='This file does not exist',
    exc=FileNotFoundError
)
```



More info - https://frappeframework.com/docs/user/en/api/py-dialog

More Dialog API (Javascript)

https://frappeframework.com/docs/user/en/api/dialog



Document API

- Given an API provided by frappe, we can use a Python console to quick check the document

  `bench --site library.test console`

- doc object represents a single row in the DB table

  `doc = frappe.get_doc('Library Member', 'xxxxx')`

  `doc.full_name`

  `doc.creation`

  `doc.owner`

More Document API - https://frappeframework.com/docs/user/en/api/document

- Example: new_doc, delete_doc, rename_doc, doc.insert, doc.save, doc.delete

Database API

- Given a database API provided by frappe, we can use a Python console to quick check the database
  - Check if a document record exists by name (True/False)

    `frappe.db.exists('Library Member', 'xxxxx')`

  - Check if record exists by filters (True/False)

```
frappe.db.exists({
    'doctype': 'Library Member',
    'first_name': 'Tony'
})
```

More DB APIs - https://frappeframework.com/docs/user/en/api/database

- Example: frappe.db.count, frappe.db.get_value, frappe.db.set_value, frappe.db.delete

Controller Validation for Membership

Back to Library Membership doctype, here is an another example of using controller hook provided by the Document class

- Objective: Ensure that when a Library membership record is created, there is not already an active membership for the Member. Need to check before submitting the document.
- Add the code (next page) into a file named library_membership.py

Controller Validation for Membership

```
class LibraryMembership(Document):
    def before_submit(self):
        exists = frappe.db.exists(
            "Library Membership",
            {
                "library_member": self.library_member,
                "docstatus": 1,                         # check for submitted documents
                "to_date": (">", self.from_date),   # check if the membership's end date is later than this membership's start date
            },
        )
        if exists:
            frappe.throw("There is an active membership for this member")         # stop the program and show a popup msg
```

Library Transaction DocType

Library Transaction

Records of an Issue or Return of an article by a Library Member who has an active membership

This doctype has the following fields:

- Article (Link) to Article
- Library Member (Link) to Library Member
- Type (Select) with 2 options: Issue and Return
- Date (Date) of transaction

It should have **Is Submittable** enabled

Controller Validation for Transaction

- Objective: When an Article is issued,
  - Verify whether the Library Member has an active membership.
  - Check whether the Article is available for Issue.
- Add the following code (next page) into a file named library_transaction.py

Controller Validation for Transaction

```
class LibraryTransaction(Document):
    def before_submit(self):
        if self.type == "Issue":
            self.validate_issue()
            # set the article status to be Issued
            article = frappe.get_doc("Article", self.article)
            article.status = "Issued"
            article.save()
        elif self.type == "Return":
            self.validate_return()
            # set the article status to be Available
            article = frappe.get_doc("Article", self.article)
            article.status = "Available"
            article.save()
    def validate_issue(self):
        self.validate_membership()
        article = frappe.get_doc("Article", self.article)
        # article cannot be issued if it is already issued
        if article.status == "Issued":
            frappe.throw("Article is already issued by another member")
```

```
def validate_return(self):
        article = frappe.get_doc("Article", self.article)
        # article cannot be returned if it is not issued first
        if article.status == "Available":
            frappe.throw("Article cannot be returned without being issued first")

def validate_membership(self):
        # check if a valid membership exist for this library member
        valid_membership = frappe.db.exists(
            "Library Membership",
            {
                "library_member": self.library_member,
                "docstatus": 1,
                "from_date": ("<", self.date),
                "to_date": (">", self.date),
            },
        )
        if not valid_membership:
            frappe.throw("The member does not have a valid membership")
```

Library Settings DocType

Library Settings

This doctype has the following fields:

- Loan Period (Int)
  - Define the loan period as number of days
- Maximum Number of Issued Articles (Int) - name: max_article
  - Restrict the maximum number of articles that can be issued by a single member

Since this doctype does not need to have multiple records, **Is Single** should be enabled

- This will not create a new DB table.
- All single values are stored in a table called **tabSingles**, usually used for storing global settings

Click **Go to Library Settings**, and then set the values for Loan Period and Maximum Number of Issued Articles

Controller Validation for Membership (Cont.)

```
class LibraryMembership(Document):
    def before_submit(self):
        exists = frappe.db.exists(
            "Library Membership",
            {
                "library_member": self.library_member,
                "docstatus": 1,                         # check for submitted documents
                "to_date": (">", self.from_date),   # check if the membership's end date is later than this membership's start date
            },
        )
        if exists:
            frappe.throw("There is an active membership for this member")         # stop the program and show a popup msg


        # get loan period from Library Settings doctype and compute to_date by adding loan_period to from_date
        loan_period = frappe.db.get_single_value("Library Settings", "loan_period")
        self.to_date = frappe.utils.add_days(self.from_date, loan_period or 30)
```

Controller Validation for Library Transaction (Cont.)

- Objective: When an Article is issued, check whether the maximum limit has reached
- Edit the following code (next page) into a file named library_transaction.py

```
def before_submit(self):
    if self.type == "Issue":
        self.validate_issue()
        self.validate_maximum_limit()
        # set the article status to be Issued
        article = frappe.get_doc("Article", self.article)
        article.status == "Issued"
        article.save()
        ....
```

```
def validate_maximum_limit(self):
    max_articles = frappe.db.get_single_value("Library Settings", "max_articles")
    count = frappe.db.count(
        "Library Transaction",
        {"library_member": self.library_member,
         "type": "Issue",
         "docstatus": 1},
    )
    if count >= max_articles:
        frappe.throw("Maximum limit reached for issuing articles")
```

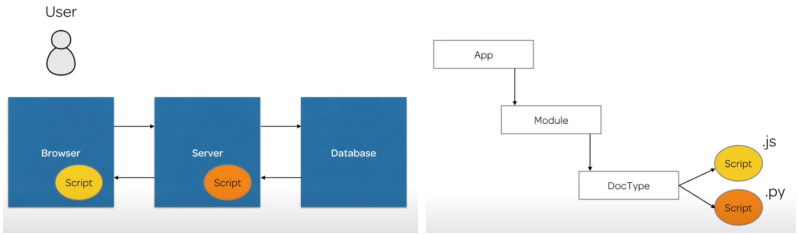
- frappe.db.count counts the number of transactions made by the member

Form Scripts

Manual or Automatic

- To create a membership for a member, the manual way is go to the Library Membership list, create a new form, select the member and other fields and then save
- Similarly, to create a transaction against a member, the manual way is go to make a new Library Transaction form
- However, this process can be automatic by using **Form Scripts**
- Form Scripts allows adding client side logic to Forms
    - Form Scripts can be written to automatically fetching values, adding validation or adding contextual actions to the form

Server-side and Client-side Scripting



Standard Form Scripts

- When a new DocType is created, a {doctype}.js is generated where custom form scripts can be written
- For example, **library_member.js** is located at frappe-bench/apps/library_management/library_management/library_management/doctype/library_member/

Syntax:

```
frappe.ui.form.on(doctype, {
    event1() {
        // handle event 1
    },
    event2() {
        // handle event 2
    }
})
```

Form Events

- Form Scripts depend on events to trigger

Syntax:

```
frappe.ui.form.on(doctype, {
    setup(frm) {
        // write handle event code
    }
})
```

- Form API - https://frappeframework.com/docs/user/en/api/form#form-api

Event Name	Description
setup	Triggered once when the form is created for the first time
before_load	Triggered before the form is about to load
onload	Triggered when the form is loaded and is about to render
refresh	Triggered when the form is loaded and rendered.
onload_post_render	Triggered after the form is loaded and rendered
validate	Triggered before before_save
before_save	Triggered before save is called
after_save	Triggered after form is saved
before_submit	Triggered before submit is called
on_submit	Triggered after form is submitted
before_cancel	Triggered before cancel is called
after_cancel	Triggered after form is cancelled
timeline_refresh	Triggered after form timeline is rendered
{fieldname}_on_form_rendered	Triggered when a row is opened as a form in a Table field
{fieldname}	Triggered when the value of fieldname is changed

New Library Membership and Transaction

Go to **library_member.js**, add the following code:

```
frappe.ui.form.on('Library Member', {
    refresh: function(frm) {
        frm.add_custom_button('Create Membership', () => {
            frappe.new_doc('Library Membership', {
                library_member: frm.doc.name
            })
        })
        frm.add_custom_button('Create Transaction', () => {
            frappe.new_doc('Library Transaction', {
                library_member: frm.doc.name
            })
        })
    }
});
```

More Info -
https://frappeframework.com/docs/user/en/api/dialog#frappenew_doc
https://frappeframework.com/docs/user/en/guides/app-development/adding-custom-button-to-form

New Library Membership and Transaction

Refresh the page and go to the Library Member form

Two buttons should appear on the top right

Click either of them will set the Library Member in each of the docs

Portal Pages

Portal Pages

Server rendered pages for website visitors

Objective: Library Members can view available Articles that they can issue from the website

To enable web views for Article doctype, Go to Article doctype and to Web View section

1. Enable **Has Web View** and **Allow Guest to View**
2. Enter **articles** in the Route field
3. Add a field named **Route** in the fields table
4. Click on Save

Article's Web View

A link **See on Website** at the top left of the form should appear

When a doctype has a webview, two HTML files are created namely:

[doctype].html and [doctype]_row.html

- Frappe uses Bootstrap 4 by default (HTML and CSS design templates)

Edit **article.html** and add the following HTML

Customise Web View Template

```
{%  extends "templates/web.html" %}

{% block page_content %}
<div class="py-20 row">
    <div class="col-sm-2">
        <img src="{{ image }}" alt="{{ title }}">
    </div>
    <div class="col">
        <h1>{{ title }}</h1>
        <p class="lead">By {{ author }}</p>
        <div>
            {%- if status == 'Available' -%}
            <span class="badge badge-success">Available</span>
            {%- elif status == 'Issued' -%}
            <span class="badge badge-primary">Issued</span>
            {%- endif -%}
        </div>
        <div class="mt-4">
            <div>Publisher: <strong>{{ publisher }}</strong></div>
            <div>ISBN: <strong>{{ isbn }}</strong></div>
        </div>
        <p>{{ description }}</p>
    </div>
</div>
{% endblock %}
```

Customising List of Articles

Open http://library.test:8000/articles to show the list of articles

Edit the **article_row.html** and add the following HTML:

```
<div class="py-8 row">
    <div class="col-sm-1">
        <img src="{{ doc.image }}" alt="{{ doc.name }}">
    </div>
    <div class="col">
        <a class="font-size-lg" href="{{ doc.route }}">{{ doc.name }}</a>
        <p class="text-muted">By {{ doc.author }}</p>
    </div>
</div>
```

More detail on Bootstrap 4: https://www.w3schools.com/bootstrap4/bootstrap_grid_basic.asp

Assignment #3

Create a Web View for Library Member and List of Library Member

- Add your name as one of the Library Member and submit your screenshot by Monday 01/02 before noon