ChatGPT

Building a Food Delivery App in Frappe

Introduction to Frappe Framework

Frappe is a full-stack web framework that uses **DocTypes** as its core building blocks. A DocType defines the model and view for a data record (like a database table and its form) [1]. For example, creating a DocType generates a JSON metadata and the underlying database table [1]. Fields in a DocType can be of various types (Data, Link, Select, etc.), enabling rich structures. Key features include:

- **DocType**: Defines the schema of a record. It can have fields (columns) of various types (e.g. Data, Link, Image). Creating a DocType automatically creates a corresponding database table [1].
- **Module**: A container for related DocTypes (e.g. a "Food Delivery" module) – it groups DocTypes in the UI.
- **Link Field**: Like a foreign key, it links one DocType to another. When you add a Link field to DocType A that points to DocType B, the user can select a record of B in A's form [2]. Internally this creates a foreign-key relationship [2].
- **Dynamic Link**: A special Link where the target DocType is chosen dynamically (based on another field). For instance, if you have two DocTypes (Customer, Delivery Partner) and a "User Type" select field, a Dynamic Link can let the user first choose "Customer" or "Delivery Partner" and then pick a record from that type [3].

Frappe also provides **Web Forms** (for building public web forms linked to DocTypes), **Server Scripts** (Python hooks on document events), **Client Scripts** (JavaScript hooks in forms), **Notifications** (email/SMS triggers), **Reports**, **Print Formats**, **Views** (Kanban, Map, List), **Workspaces/Dashboards**, and a robust **Role-based permission** system.

Below is a step-by-step guide to building a Food Delivery app covering all these concepts.

Data Model and DocTypes

We will create these main DocTypes: **Region**, **Restaurant**, **Food Item**, **Delivery Partner**, and **Food Order**. Their relationships will be:

- **Region**: Simple DocType (e.g. name of city/area).
- **Restaurant**: Fields: Name (Data), Region (Link→Region), Address (Data), Logo (Image), Manager (Data or Link to User), etc. (Each restaurant belongs to a region.)
- **Food Item**: Fields: Name (Data), Price (Currency), Restaurant (Link→Restaurant), Image (Image), etc. (Each food item is offered by one restaurant.)
- **Delivery Partner**: Fields: Name (Data), Region (Link→Region), Contact (Data), etc. (Each partner is based in one region.)
- **Food Order**: Fields: Customer (Link→User or Website User), Food Item (Link→Food Item), Restaurant (Dynamic Link or Link→Restaurant, auto-filled), Region (Link→Region, auto-filled), Price (Currency,

auto-filled), Status (Select: e.g. “Ordered”, “Cooking”, “On the Way”, “Delivered”), Delivery Partner (Link→Delivery Partner, assigned by script).

The Entity-Relationship diagram looks like this:

【embed_image】 *Figure: Entities and relationships in the Food Delivery app (Restaurant→Region, Food Item→Restaurant, Delivery Partner→Region, Food Order links all together).*

- A **Region** 1– *Restaurants**. Each Restaurant links to one Region via a Link field [2].
- A **Restaurant** 1– *Food Items**. Each Food Item links to its Restaurant.
- A **Region** 1– *Delivery Partners**. Each partner has a Link to the Region they cover.
- A **Food Order** links to one Customer, one Food Item, one Restaurant, one Region, and one Delivery Partner. The restaurant and region fields can be fetched (auto-filled) from the selected food item (since the item “knows” its restaurant and region). The Delivery Partner is assigned by script after order creation.

By using Link fields, Frappe automatically provides dropdown selection of related records. Dynamic Link fields would be used if the linked DocType depends on another field’s value. In our simple model we use plain Links since each field’s DocType is fixed [2].

Creating the DocTypes

Using Frappe’s **DocType builder**, create each DocType under a new Module (e.g. “Food Delivery”). For each DocType, define the fields as follows (here “Options” denotes target DocType for Link fields):

- **Region**:

- *region_name* (Data).

- **Restaurant**:

- *restaurant_name* (Data) – set as Title Field.
- *region* (Link, Options: “Region”) – to select the Region.
- *address* (Data).
- *manager* (Data) – the name or email of the manager. (We will use this later for Notifications.)
- *logo* (Attach Image).
- *rating* (Float).

- *status* (Select: “Pending”, “Approved”, “Rejected”) – used with Workflow (see below).

- **Food Item**:

- *food_name* (Data) – Title Field.
- *restaurant* (Link, Options: “Restaurant”).
- *price* (Currency).
- *image* (Attach Image).

- *(any other fields like description).*

- **Delivery Partner**:

- *partner_name* (Data).

- *region* (Link, Options: “Region”).

- **Food Order**:

- *customer* (Link, Options: “User” or “Customer” DocType) – the person placing the order.
- *food_item* (Link, Options: “Food Item”).
- *price* (Currency).
- *restaurant* (Link, Options: “Restaurant”).
- *region* (Link, Options: “Region”).
- *status* (Select: e.g. “Ordered”, “Cooking”, “On the Way”, “Delivered”).
- *delivery_partner* (Link, Options: “Delivery Partner”).

In the **Restaurant** DocType’s fields setup, note the “Links” section: once you set Restaurant → Region and Restaurant → Food Item (as above), Frappe will know the relationships. The transcript explains that a Link is essentially a foreign key linking one record to another [2].

Likewise, you can use a **Fetch From** property to auto-fill fields: for example, in **Food Order**, set *price* to fetch from `food_item.price`, set *restaurant* to fetch from `food_item.restaurant`, and *region* to fetch from `restaurant.region`. Then when a user selects a Food Item, the related price/restaurant/region will auto-populate in the order form [4].

Linking DocTypes (Link vs Dynamic Link)

As mentioned, **Link** fields create fixed relationships. When the user clicks into a Link field, they can search/select any record of that DocType [2]. In contrast, a **Dynamic Link** field requires the user to first choose a document type. For example, if you wanted one field that could link either to a Customer or a Delivery Partner based on a preceding “User Type” select, you would use Dynamic Link. The transcript explains that with a Dynamic Link, if the user selects “Customer” as the type, the field shows Customer records, and if “Delivery Partner” is selected, it shows Delivery Partners [3]. We don’t strictly need that here (we use separate fields and roles), but keep it in mind for complex cases.

Setting Up Web Forms

To let users (customers) place orders through the website, create a **Web Form** for the *Food Order* DocType. In Frappe’s Desk, search for “Web Form” and make a New Web Form: select DocType = **Food Order**. Then configure:

- **Form Fields**: Click *Get Fields* to import Food Order fields. Show only those users should see (e.g. Food Item, maybe quantity if added). Hide fields that should not be changed by users. For example, hide *customer* (since we’ll auto-fill it), *restaurant*, *region*, and *delivery_partner* (these will be auto-filled). The transcript shows hiding fields to avoid confusion: “you don’t want to show customer because they are logged in…” [5].

- **Permissions**: Check "Login Required" so only authenticated users can order. Check "Auto Select" for the customer field so it uses the session user (e.g. website user's email) for *customer*. In the Web Form settings enable "Show List" – this allows the user to see all orders they've placed. In the transcript demo, the instructor turned on "Only logged in user" and saw the order record appear in the list after submission [5] [6].
- **Button/CTA**: Set the button label (e.g. "Place Order"). Optionally add a banner or success message. You can also redirect to a URL after submit.

When a user visits the web page (e.g. /order-food), they see the Web Form. Thanks to **Hidden** and **Fetch From** settings, the form will automatically fill in *customer* = their user, and fetch price/restaurant/region from the selected food item [4] [5]. After submission, they might see a confirmation and then the record appears under "My Orders".

To **prefill fields via URL**, use query parameters. For example, linking to the form as `/order-food/new?food_item=Pizza` will prefill the Food Item field with "Pizza" [7]. Frappe supports passing `?fieldname=value` in the URL to auto-fill that field [7]. We'll use this below when setting up the menu page.

Server Scripts (Business Logic)

Frappe **Server Scripts** allow custom Python code on DocType events [8]. For example, when a *Food Order* is created, we want to auto-assign a delivery partner. We can add a Server Script of type *"Document Event"* on Food Order, event *"Before Insert"* [9]. In that script, write something like:

```
# Server Script: Before Insert on Food Order
# Assign a delivery partner based on region
region = doc.region
partner = frappe.get_list("Delivery Partner",
    filters={"region": region, "status":"Active"},
    fields=["name"], limit=1)
if partner:
    doc.delivery_partner = partner[0].name
```

This gets the Food Order's region (inherited from the food item's restaurant) and queries the Delivery Partner list for one in that region. It then sets the `delivery_partner` field. (The transcript demo showed this logic: on before_insert, get the region and assign the partner automatically [10].)

You can also use Server Scripts to enforce business rules (e.g. validate fields) or trigger actions on Save/Submit. Server Scripts run on the server for security, whereas **Client Scripts** (next section) run in the browser [9].

Notification Rules and Email Templates

Frappe **Notifications** (Email Alerts) let you email users when events occur [11]. We'll set up alerts so that when a new Food Order is created, we notify the restaurant manager and/or delivery partner. In the Desk, go to *Setup* → *Notification* (or Email Alert):

- **Send Alert On:** New (so it triggers when a new record is made).
- **Document Type:** Food Order.
- **Recipients:** Choose "Document Email Address" and pick *restaurant_manager* (assuming the Restaurant has a field *manager* for the email). This means the Notification will send to the email in that field. You could also add a fixed email or other fields.
- **Subject/Message:** You can use Jinja templating with `{{ }}` to insert fields [11]. For example, set Subject: `New Order {{ doc.name }}` or `Order for {{ frappe.db.get_value("Food Item", doc.food_item, "food_name") }}`. In the body, you might include details:

```
<h3>New Order {{ doc.name }}</h3>
<p>Food Item: {{ frappe.get_value("Food Item", doc.food_item, "food_name") }}</
p>
<p>Customer: {{ doc.customer }}</p>
<p>Quantity: {{ doc.quantity or 1 }}</p>
<p>Price: {{ doc.price }}</p>
<p>Status: {{ doc.status }}</p>
```

Under the hood, `doc.[fieldname]` works for simple fields. If you need linked data, use `frappe.get_value` or `frappe.get_doc` in Jinja to fetch that field value. For instance, the instructor discovered that using `{{ doc.food_item }}` gave just the ID, so he used Jinja to fetch the food's name via its DocType [12]. E.g.: `{{ frappe.get_value("Food Item", doc.food_item, "food_name") }}`.

Finally save the Notification. When a new order is placed, the restaurant's manager (email) will receive a formatted email. You could similarly notify the Delivery Partner (if you store their email) by adding another Notification rule for Food Order creation with recipient = `delivery_partner.user` or similar.

Food Item Listing Page (Frappe Builder)

To let customers browse and select food items, we'll build a public **Web Page** using *Frappe Builder* (or the Website Pages feature). The page will dynamically list all Food Items in a card layout. Steps:

1. **Create Web Page:** In the Desk, ensure Frappe Builder is installed. Go to Website > Web Page (or use the Builder) and create a new page "Food Menu" or similar.
2. **Data Script:** In the page's Data section, write a small Python script to fetch all food items:

```
data.items = frappe.db.get_all("Food Item",
    fields=["name", "food_name", "price", "image", "restaurant"])
```

(This uses `frappe.db.get_all` to retrieve records [13]. For example code, see Frappe docs: `data.users = frappe.db.get_all("User")` [13].)

3. **Page Layout:** Add a **Repeater** block that loops over `data.items`. Inside, design a card layout: show the image, then the food name (`{{ item.food_name }}`), price, and an "Order" button. Bind the dynamic fields (image, name, price) from `data.items`.
4. **Order Button:** Set the Order button's URL to the Food Order web form, pre-filling the `food_item`. For example: `/order-food/new?food_item={{ item.name }}`. This passes the food item's name (ID) in the query so the web form opens with that item selected [7] [14]. (One can also prefill price via URL, but better rely on server fetch for security.) The transcript demo did exactly this: each card's "Order" button linked to `/order/new?food_item=...`, allowing instant ordering [14] [7].
5. **Styling:** Use the builder's design tools (CSS in the builder or custom CSS) to make the cards grid, spacing, etc. The transcript showed adjusting card style, centering, padding, etc, until all items appear nicely [15] [16].
6. **Publish:** Save and publish the page. Now the site has a menu page with all food items and working order links.

The key is dynamic data binding: `data.items = frappe.db.get_all("Food Item")` [13] and using `{{ item.field }}` in the page template. This makes the page update automatically as items are added.

Workflow: Restaurant Approval

Suppose restaurants must be approved before accepting orders. Frappe Workflows let you set states and transitions. In Restaurant DocType, enable **Workflow** and add a Workflow for "Restaurant Onboarding":
- States: *Draft* (new restaurant record), *Submitted* (submitted for review), *Approved*, *Rejected*.
- Transitions: From **Draft** user can *Submit* (to go to Submitted), from **Submitted** an Admin can *Approve* or *Reject*.
- Actions: On *Approve*, state becomes *Approved*; on *Reject*, state *Rejected*. Only Admin users have the right to approve/reject.

【embed_image】 *Figure: Workflow states for Restaurant approval (Draft → Submitted → Approved or Rejected).*

By configuring Workflow in the DocType settings, the Restaurant form will show action buttons (Submit, Approve, Reject) and enforce that orders can only be placed if `status == "Approved"`.

Reports (Query and Script Reports)

For analytics, create custom reports:

- **Query Report**: Uses SQL. For example, a report "Most Ordered Foods" on the *Food Order* DocType might use a query like:

```
SELECT `tabFood Item`.food_name AS Item, COUNT(`tabFood Order`.name) AS
Orders, SUM(`tabFood Order`.price) AS Total
  FROM `tabFood Order`
```

```
JOIN `tabFood Item` ON `tabFood Order`.food_item = `tabFood Item`.name
GROUP BY `tabFood Order`.food_item;
```

This SQL selects food names and counts orders/prices. Frappe docs explain making a Query Report by writing such an SQL query [17]. In the Report Builder, set Type = "Query Report" and paste the SQL into the "Query" field [17].

- **Script Report**: Uses Python. For more complex logic, make a Script Report. For example, a "Orders by Region" report might use Python to aggregate:

```
# In <app>/reports/orders_by_region/orders_by_region.py
def execute(filters=None):
    columns = ["Region:Data:100", "Orders:Int:100", "Revenue:Currency:100"]
    data = []
    region_list = frappe.get_all("Region", fields=["name"])
    for r in region_list:
        orders = frappe.get_all("Food Order",
            filters={"region": r.name},
            fields=["COUNT(name) as count", "SUM(price) as total"])
        if orders:
            data.append([r.name, orders[0].count or 0, orders[0].total or 0])
    return columns, data
```

Alternatively, in a custom report (enabled in Developer Mode), you can use `frappe.db.get_all` directly (as shown in Frappe docs) [18]. For example: `return frappe.db.get_all('User', ['first_name','last_name'], filters=filters)` is given as a sample [18].

Use the Role Permissions Manager to allow only Admin/Manager roles to view these reports. Reports can be linked in the Workspace later.

Print Format & Automated Email

You can design custom Print Formats for orders (invoices/receipts). In Print Format Builder, drag-drop fields or write HTML/Jinja. Frappe uses Jinja in print templates [19]. For example, an HTML Print Format for an order could include:

```
<h3>Order {{ doc.name }}</h3>
<p>Customer: {{ doc.customer }}</p>
<p>Date: {{ doc.order_date }}</p>
<table>
<tr><th>Item</th><th>Qty</th><th>Rate</th><th>Amount</th></tr>
{% for row in doc.items %}
<tr>
  <td>{{ row.item_name }}</td>
```

```
  <td>{{ row.qty }}</td>
  <td>{{ row.rate }}</td>
  <td>{{ row.amount }}</td>
</tr>
{% endfor %}
</table>
```

This uses `{{ doc.field }}` to insert data [19]. The docs provide an example for an Invoice template [20].

For **automated emails** on delivery, you could:
- Set a Notification to trigger on *Save* or *Submit* of Food Order with status changed to "Delivered". The recipient would be the *customer* (link to email) or any role. The message can attach a PDF by using the Print Format.
- Or use a Server Script on Food Order *After Save* (status == "Delivered") to call `frappe.sendmail` with the print format attached.

Frappe's built-in Notifications (email alerts) handle attaching PDFs if configured.

Views: Kanban, Map, Workspace, Dashboard

- **Kanban View**: Frappe's Kanban Board lets you visualize records by status [21]. You can make a Kanban board for *Food Order* (or *Restaurant* approvals), grouping by status (Ordered/Cooking/On the Way/Delivered). Cards representing orders can be dragged as their status changes [21].
- **Map View**: If you add a *Geolocation* field (or use address), Frappe can show a Map view of records. For example, you could map Delivery Partner locations or Restaurant addresses. (Add a Geolocation field and Frappe will show the records on a map automatically.)
- **Workspace and Dashboards**: Create a custom Workspace for the Food Delivery app. In the Workspace, add Quick Links (e.g. "New Order", "Restaurant List"), Kanban boards, and Report charts. You can add Dashboard Cards (via HTML or Frappe Charts) showing metrics like "Pending Orders" count, "Weekly Sales", etc. This involves using Frappe's chart or widget API.
- **Example Dashboard Card**: A simple card might use a Client Script or Data Script to count pending orders and display that number. For instance, on page load:

  ```
  frappe.db.count('Food Order', {status:'Ordered'}).then(r => {
      frappe.msgprint(r); // or set in the UI
  });
  ```

  This uses the `frappe.db.count` API on the client side (as demoed in the transcript) to fetch how many orders have status "Ordered" [22].

Role-Based Permissions

In Frappe's **Role Permissions Manager**, assign which roles can do what on each DocType [23]. For example:

- Restaurant: Only "Manager" or "System Manager" roles have Write/Create; all logged-in users (role "Guest" or "All Users") can create a Web Form order if permission allows.
- Food Item: Only Restaurant admins or Manager role can create; Viewer roles can read.
- Food Order: Customer role can create (via web form) and read their own orders; Restaurant Manager role can read orders for *their* restaurant; Delivery Partner role can read orders assigned to them.
- Delivery Partner: Only Delivery Partner role or admin can view.

In the DocType's "Permissions" table, add rows linking Roles → Permissions (Read/Write/Create). The docs explain adding default roles in the Permissions table [23]. Set appropriate **Permission Level** or **User Permissions** if needed (e.g. restrict a Delivery Partner to only see orders where they are assigned, using user permission on the dynamic link).

Client Scripts

Client Scripts (JavaScript) customize form behavior in the browser [24]. Examples:
- **Fetch on change**: Automatically fill fields when one changes. E.g., fetch price when Food Item is selected:

```
frappe.ui.form.on('Food Order', {
  food_item: function(frm) {
    frappe.db.get_value('Food Item', frm.doc.food_item, 'price')
      .then(r => frm.set_value('price', r.message.price));
  }
});
```

Or use the shortcut `cur_frm.add_fetch('food_item', 'price', 'price');` to auto-fetch the price field. (Frappe docs give a similar example for customer tax number [25].)
- **Form Validation**: On form *Validate* or *Before Save*, add checks. For instance, prevent ordering more than stock, or ensure status moves correctly. Example from docs (on Task dates) [26] shows how to use `frappe.ui.form.on('Task','validate', ...)`. - **Custom Button/Indicator**: Add custom buttons or info. For instance, on the Food Order list, you could add an action button "Notify Restaurant" that calls an API. The transcript even shows adding buttons in List view with conditions [27].
- **Pending Orders Count**: As per the requirement, you might display "N pending orders" on the dashboard. A client script on the workspace page could use `frappe.db.count('Food Order', {status: 'Ordered'})` [22] and then display the number (e.g. by setting inner text of a HTML container).
- **Form Indicators**: Change form color or header based on status: e.g., if order is "Delayed", show a red indicator:

```
frappe.ui.form.on('Food Order', {
  refresh: function(frm) {
    if (frm.doc.status=='Delayed') {
      frappe.set_route("Form", "Color Indicator", frm.doctype, frm.docname);
```

```
    }
  }
});
```

(Frappe supports callouts and indicators via JS APIs.)

Frappe's client scripting is very flexible [24] [26], allowing any JavaScript logic on the form. It complements server scripts for a rich user interface.

Sources: This guide is based on Frappe Framework documentation and the provided implementation transcript. Key references include Frappe's official docs on DocTypes [1], Notifications [11], Data Scripts [13], Server Scripts [8] [9], Script Reports [18], Print Formats [19], Kanban view [21], and Permissions [23]. The transcript from the video was used to capture the specific app-building steps.

---

[1] Understanding DocTypes
https://docs.frappe.io/framework/user/en/basics/doctypes

[2] [3] [4] [5] [6] [10] [12] [14] [15] [16] [22] [27] ts.txt
file://file-5F9ekWKmTGrAaN1XauH1cg

[7] Prefill web form fields - Website - Frappe Forum
https://discuss.frappe.io/t/prefill-web-form-fields/27096

[8] [9] Server Script
https://docs.frappe.io/framework/user/en/desk/scripting/server-script

[11] Notification
https://docs.frappe.io/erpnext/user/manual/en/notifications

[13] Data Script
https://docs.frappe.io/builder/data-script

[17] How To Make Query Report
https://docs.frappe.io/framework/user/en/guides/reports-and-printing/how-to-make-query-report

[18] Script Report
https://docs.frappe.io/framework/user/en/desk/reports/script-report

[19] [20] Customize Print Format
https://docs.frappe.io/erpnext/user/manual/en/records-print-format

[21] Kanban Board
https://docs.frappe.io/erpnext/user/manual/en/kanban-board

[23] Users and Permissions
https://docs.frappe.io/framework/user/en/basics/users-and-permissions

[24] [25] [26] Client Script
https://docs.frappe.io/framework/user/en/desk/scripting/client-script