Day2

**Planning the**

**Technical Foundation**

Shipment Tracking API

Fronted Next.js

# ***user***

User: Performs action like browsing product or placing an order.

Fronted (Next.js): Send user action to tha backend and third-party APIs.

Sanity (CMS): Manges and store all order and product data.

Third Party APIs: handle shipment tracking and payment processing.

Shipment Tracking: provide real-time delivery update to the frontend.

Payment Gateway: processes payment and send confirmation back..

Order Data: Stored in Sanity CMS for backend reference.

**User Registration:**

Seller Registration

Buyer Registration

**Product Listing (Seller Side):**

Seller adds a product

Provides product details (Title, Description, Price, Images,size,color).

Admin approval (Optional)

**Browsing Products (Buyer Side):**

Buyer searches for products

Uses filters and categories

**Product Selection:**

Opens the product detail page

Checks reviews and ratings

**Add to Cart:**

Adds the product to the cart

Option to add multiple products

**Checkout** **Process**:

Adds shipping details

Selects payment method

Places the order

**Order Confirmation:**

Seller gets notified

Order packing and shipment begin

**Delivery:**

Tracking details are shared with the buyer

Buyer receives the order

**Feedback System:**

Buyer provides product ratings and reviews

**API Integration:**

**API integration refers to the process of connecting two or more applications or systems via their Application Programming Interfaces (APIs) to enable them to exchange data and functionalities seamlessly. It allows disparate systems to work together, automating processes and improving efficiency by eliminating the need for manual intervention.**

**Api Schema Example:**

export default {

name: 'contact',

title: 'Contact',

type: 'document',

fields: [

{

name: 'name',

title: 'Name',

type: 'string',

validation: (Rule) => Rule.required(),

},

{

name: 'email',

title: 'Email',

type: 'email',

validation: (Rule) =>

Rule.required().error('A valid email address is required'),

},

{

name: 'phone',

title: 'Phone Number',

type: 'string',

},

{

name: 'address',

title: 'Address',

type: 'text',

},

{

name: 'company',

title: 'Company',

type: 'reference',

to: [{ type: 'company' }],

},

{

name: 'tags',

title: 'Tags',

type: 'array',

of: [{ type: 'reference', to: [{ type: 'tag' }] }],

},

{

name: 'notes',

title: 'Notes',

type: 'text',

},

{

name: 'profilePicture',

title: 'Profile Picture',

type: 'image,

},

],

};

Schema design:

Schema design refers to the process of organizing and structuring the data in a database or application to ensure it is efficient, scalable, and easy to manage. It involves defining the framework that dictates how data is stored, accessed, and related within a system.

### Key Components of Schema Design:

1. \*\*Entities and Tables\*\*:

Identify the entities (objects) that the system will track, such as users, products, or orders, and represent them as tables in a relational database or collections in a NoSQL database.

2. \*\*Attributes and Fields\*\*:

Define the attributes (properties) for each entity, which become columns (fields) in the database. For example, a `User` table may include `id`, `name`, `email`, and `created\_at`.

3. \*\*Data Types\*\*:

Specify the type of data each field will store, such as integers, strings, dates, or JSON objects.

4. \*\*Relationships\*\*:

Establish how entities are related. In a relational database, these are represented by foreign keys. Relationships can be one-to-one, one-to-many, or many-to-many.

5. \*\*Constraints and Rules\*\*:

Define rules to ensure data integrity, such as primary keys, unique constraints, foreign key constraints, and validation rules.

6. \*\*Indexes\*\*:

Design indexes to optimize query performance by enabling faster lookups on frequently queried fields.

### Types of Schemas:

1. \*\*Logical Schema\*\*:

High-level design that focuses on entities and relationships without considering the specific database implementation.

2. \*\*Physical Schema\*\*:

Detailed schema implementation tailored to the chosen database system, including storage formats and indexing strategies.

### Example Schema Design:

For an \*\*e-commerce website\*\*, the schema might look like this:

- \*\*Users Table\*\*:

- `id` (Primary Key)

- `name`

- `email` (Unique)

- `password`

- `created\_at`

- \*\*Products Table\*\*:

- `id` (Primary Key)

- `name`

- `description`

- `price`

- `stock\_quantity`

- `created\_at`

- \*\*Orders Table\*\*:

- `id` (Primary Key)

- `user\_id` (Foreign Key referencing Users)

- `total\_price`

- `status`

- `created\_at`

- \*\*Order\_Items Table\*\*:

- `id` (Primary Key)

- `order\_id` (Foreign Key referencing Orders)

- `product\_id` (Foreign Key referencing Products)

- `quantity`

- `price\_per\_unit`

Proper schema design ensures your database operates efficiently and reliably, even as your application grows.