# Technical Foundation Plan for Insight Artistry

## 1. Define Technical Requirements

### Frontend Requirements:

- A responsive interface that includes:
  - Home Page: Displays featured artworks, categories, and promotions.
  - Product Listing Page: Allows filtering by categories (e.g., Paintings, Sculptures) and price range.
  - Product Details Page: Displays artwork details, customization options, and related items.
  - Cart and Checkout Pages: Simplify the purchasing process with real-time updates.
  - User Profile Page: Shows order history, customization requests, and saved items.

### Backend Requirements:

- **Database:** Firebase for storing:
  - Products (details, stock levels, and images).
  - Customers (registration details, preferences).
  - Orders (status, payment history).
- **APIs:** Integration for:
  - Payment processing (Stripe, PayPal).
  - Shipping updates (Third-party API).
  - Customization management.

### Third-Party APIs:

- Payment Gateway API for secure transactions.
- Shipping API for real-time delivery tracking.
- Customization API to process user modification requests.

## 2. Design System Architecture

### High-Level Architecture Diagram:

```
[Frontend (Next.JS)]
    |
[Firebase Database] <--> [Third-Party APIs]
    |                  |
[Authentication]    [Payment Gateway, Shipping API]
```

**Workflows:**

1. **User Registration:**
    - **Input:** User details.
    - **Action:** Data stored in Firebase.
    - **Output:** Account creation and login credentials.
2. **Product Browsing:**
    - **Input:** User selects a category or applies filters.
    - **Action:** Firebase query fetches matching artworks.
    - **Output:** Artworks displayed dynamically.
3. **Order Placement:**
    - **Input:** Items added to cart and checkout initiated.
    - **Action:** Firebase records order details; payment API processes transaction.
    - **Output:** Order confirmation sent to user.
4. **Customization Request:**
    - **Input:** User submits customization details.
    - **Action:** Request logged in Firebase; admin notified for approval.
    - **Output:** Cost estimate shared with user.
5. **Shipping Updates:**
    - **Input:** Order ID queried via Shipping API.
    - **Action:** Fetch real-time status.
    - **Output:** Delivery updates displayed in the user profile.

## 3. Plan API Requirements

### Endpoints:

1. **Fetch Products:**
    - Endpoint: /products
    - Method: GET
    - **Response:**

```
{
  "id": "1",
  "name": "Abstract Painting",
  "price": 500,
  "category": "Painting",
  "stock": 10,
  "image": "url"
}
```

2. **Place Order:**

- Endpoint: /orders
- Method: POST

**Payload:**

```
{
  "userId": "123",
  "products": [{"id": "1", "quantity": 2}],
  "paymentStatus": "Paid"
```

}

**Response:**
`{"orderId": "456", "status": "Confirmed"}`

3. **Track Shipment:**
- Endpoint: /shipment
- Method: GET
- **Response:**

```
{
  "orderId": "456",
  "status": "In Transit",
  "eta": "3 days"
}
```

4. **Customization Request:**
- Endpoint: /customization
- Method: POST
- **Payload:**

```
{
  "productId": "123",
  "size": "24x36",
  "material": "Canvas",
  "colorScheme": "Black and White"
}
```

**Response:**
{"requestId": "789", "status": "Received"}

## 4. Write Technical Documentation

### System Architecture Overview:

- Frontend: Built using Nest.JS for a responsive, user-friendly interface.
- Backend: Firebase database for real-time data synchronization.
- Third-Party APIs: Integrated for payment processing, shipping updates, and customization handling.

**Key Workflows:**

1. User Onboarding:
   - New users register, and data is stored in Firebase.
   - User accounts enable browsing, purchasing, and customization.
2. Order Management:

    ○ Products are added to the cart and processed at checkout.
    ○ Payment confirmation triggers order recording and shipping.
3. Customization Handling:
    ○ User modifications are logged and forwarded to admins for processing.

### Sanity Schemas for Orders and Customers

Schema for Orders

```
export default {
  name: 'order',
  type: 'document',
  fields: [
    { name: 'orderId', type: 'string', title: 'Order ID' },
    { name: 'customerId', type: 'reference', to: [{ type: 'customer' }], title: 'Customer ID' },
    { name: 'products', type: 'array', of: [{ type: 'reference', to: [{ type: 'product' }] }], title:
'Products' },
    { name: 'totalAmount', type: 'number', title: 'Total Amount' },
    { name: 'paymentStatus', type: 'string', options: { list: ['Paid', 'Pending'] }, title: 'Payment
Status' },
    { name: 'orderDate', type: 'datetime', title: 'Order Date' },
    { name: 'shippingStatus', type: 'string', options: { list: ['Pending', 'Shipped', 'Delivered'] },
title: 'Shipping Status' },
  ],
};
```

Schema for Customers

```
export default {
  name: 'customer',
  type: 'document',
  fields: [
    { name: 'customerId', type: 'string', title: 'Customer ID' },
    { name: 'name', type: 'string', title: 'Full Name' },
    { name: 'email', type: 'string', title: 'Email Address' },
    { name: 'address', type: 'text', title: 'Address' },
    { name: 'phone', type: 'string', title: 'Phone Number' },
    { name: 'orderHistory', type: 'array', of: [{ type: 'reference', to: [{ type: 'order' }] }], title:
'Order History' },
  ],
};
```

## Collaboration Notes

### Collaboration Overview

During the preparation of the technical foundation document, collaboration was facilitated through brainstorming sessions and feedback reviews.

**Feedback Summary**

- **Peer Review:** Shared workflows and API designs with peers to identify potential gaps.
- **Mentor Feedback:** Received suggestions to improve the schema for `Orders` and `Customers` by adding relationships and refining attributes.
- **Incorporated Changes:** Adjusted API payloads and responses to better align with real-world use cases.

**Tools Used**

- **Communication:** Google Meet for brainstorming and feedback discussions.
- **Version Control:** GitHub for tracking schema updates and document changes.

## Updates to the System Architecture Diagram

Include the updated architecture with additional details for entities:

```
[Frontend (Next.js)]
    |
[Firebase Database] <--> [Third-Party APIs]
    |                  |
[Authentication]    [Payment Gateway, Shipping API]
    |
[Entities: Products, Orders, Customers]
```

## Updated API Endpoints

### Endpoint for Fetching Customer Details

- **Endpoint:** `/customers/{customerId}`
- **Method:** GET
- **Description:** Retrieve customer information, including order history.
- **Response Example:**

```
{
  "customerId": "123",
  "name": "John Doe",
  "email": "john.doe@example.com",
  "address": "123 Art Street, Creativity City",
  "phone": "123-456-7890",
  "orderHistory": [
    {
      "orderId": "456",
      "totalAmount": 500,
      "paymentStatus": "Paid",
      "orderDate": "2025-01-10",
      "shippingStatus": "Delivered"
    }
  ]
}
```

### Endpoint for Fetching Order Details

- **Endpoint:** `/orders/{orderId}`
- **Method:** GET
- **Description:** Retrieve details of a specific order, including customer and product information.
- **Response Example:**

```
{
  "orderId": "456",
  "customerId": "123",
  "products": [
    { "id": "1", "name": "Abstract Painting", "quantity": 1 },
    { "id": "2", "name": "Modern Sculpture", "quantity": 2 }
  ],
  "totalAmount": 1000,
  "paymentStatus": "Paid",
  "orderDate": "2025-01-10",
  "shippingStatus": "Shipped"
}
```

## Additions to Workflows

### Order Management Workflow

- **Step 1:** User selects products and adds them to the cart.
- **Step 2:** Order details are sent to Firebase and linked to the customer's profile.
- **Step 3:** Payment Gateway API processes the transaction and confirms the payment.
- **Step 4:** Shipping API updates the order status as the product is dispatched and delivered.

### Customer Profile Management

- **Step 1:** Customer registers or updates profile information.
- **Step 2:** Firebase stores customer details and links order history.
- **Step 3:** Customer accesses order history and updates via `/customers/{customerId}` endpoint.