Marketplace Technical Foundation - Furniro

Table of Contents:

By - Shafique Ur Rehman

Technical Foundation of Robiz (General Ecommerce Website)

Furniro is envisioned as a robust and user-friendly e-commerce platform that delivers a seamless shopping experience across devices. To achieve this, the technical foundation must align with the business goals, ensuring scalability, performance, and ease of use for customers and administrators. This document outlines the core technical requirements for the frontend, backend, and integrations, ensuring that Furniro meets industry standards and customer expectations.

Technical Requirements:

1. Frontend Requirements

- Nextjs for server side rendering
- Tailwind utilities for responsiveness
- Redux Toolkit for global state management

Essential Routes:

- **Home**
- **Category** - for renderd products category wise dynamically
- **Subcategory** - Mostly products have subcategory inside the catgeory
- **Cart** - this allow user to add and remove edit products
- **Checkout** - A process to capture shipping, payment details, and apply discount codes.
- **Order** - Provide order summary, tracking links, and estimated delivery timelines.
- **Login** - where user can login if they have an account already
- **Signup** - User can register itself if not already have an account
- **Profile** - Each user have its own profile after registration

Similarly there will be more pages but these are very eassetial

2. Backend Requirements

Headless cms:

- We will use Headless cms to manage product data/Catalog
- Flexible schema definitions for dynamic business needs.
- Role-based access controls for secure data management.
- Real-time collaboration features for content editing.

Authentication and Authorization

- **Auth0 or NextAuth.js:**
    - Secure user authentication with support for social logins (e.g., Google, Facebook).
    - Manage session tokens and multi-factor authentication (MFA) for additional security.

3. Third-party api

Shipping API Integration:

- Real-time shipping rates, label printing, and shipment tracking using APIs like DHL, FedEx, or Shippo.
- Delivery Notifications: Notify customers at every stage of delivery via email or SMS.

Payment Gateways:

- Integrate APIs for secure payment handling (e.g., Stripe, PayPal, Razorpay).
- Support for multiple payment options, including credit/debit cards, wallets, and UPI.

Tax and Currency Calculation:

- Automate tax computation and support multi-currency transactions using third-party APIs.

Email and SMS Notifications:

- Notify customers about order status, promotions, and updates using services like Twilio or SendGrid.

Product Search and Recommendations

- Algolia: Fast, customizable product search and filter functionality.
- Elasticsearch: Advanced search engine for handling large datasets with full-text search capabilities.
- AWS Personalize: AI-powered personalized product recommendations and search results.

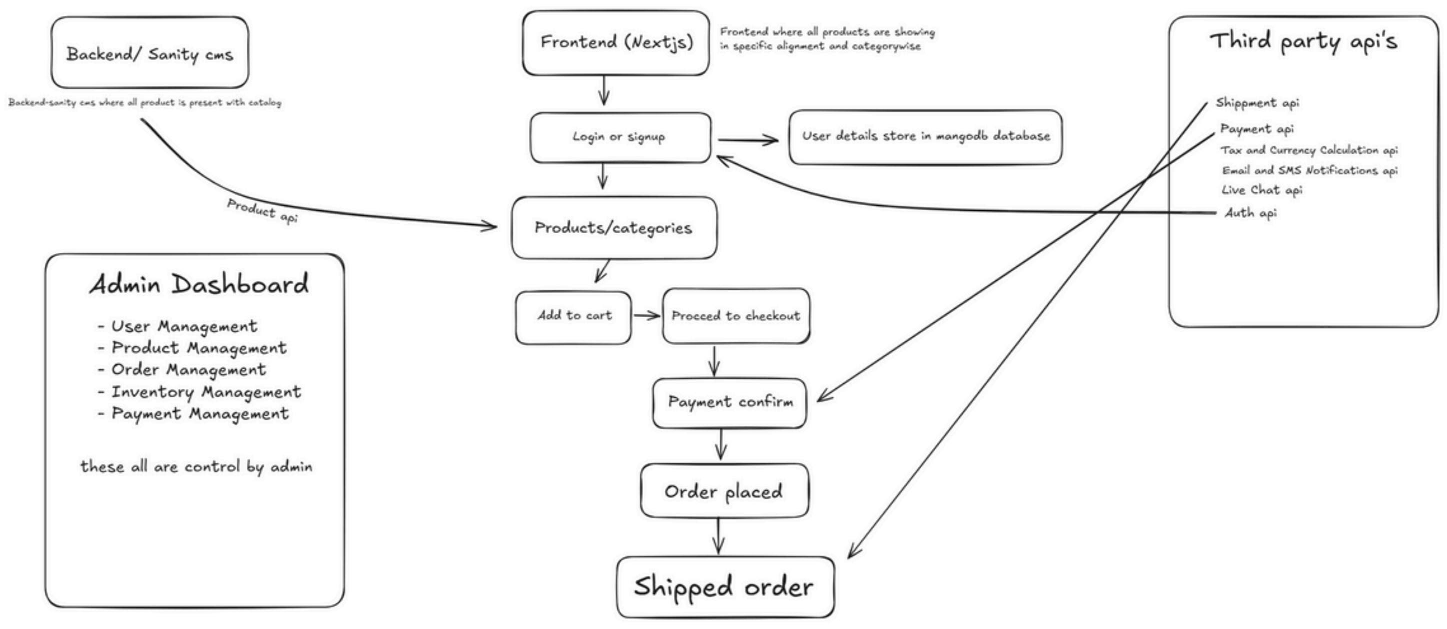
Social Media and Marketing

- Facebook Ads API: Automate and analyze ad campaigns.
- Google Ads API: Manage ad campaigns directly through integrations.
- Mailchimp API: Email marketing campaigns and automated customer engagement workflows.

Live Chat and Support

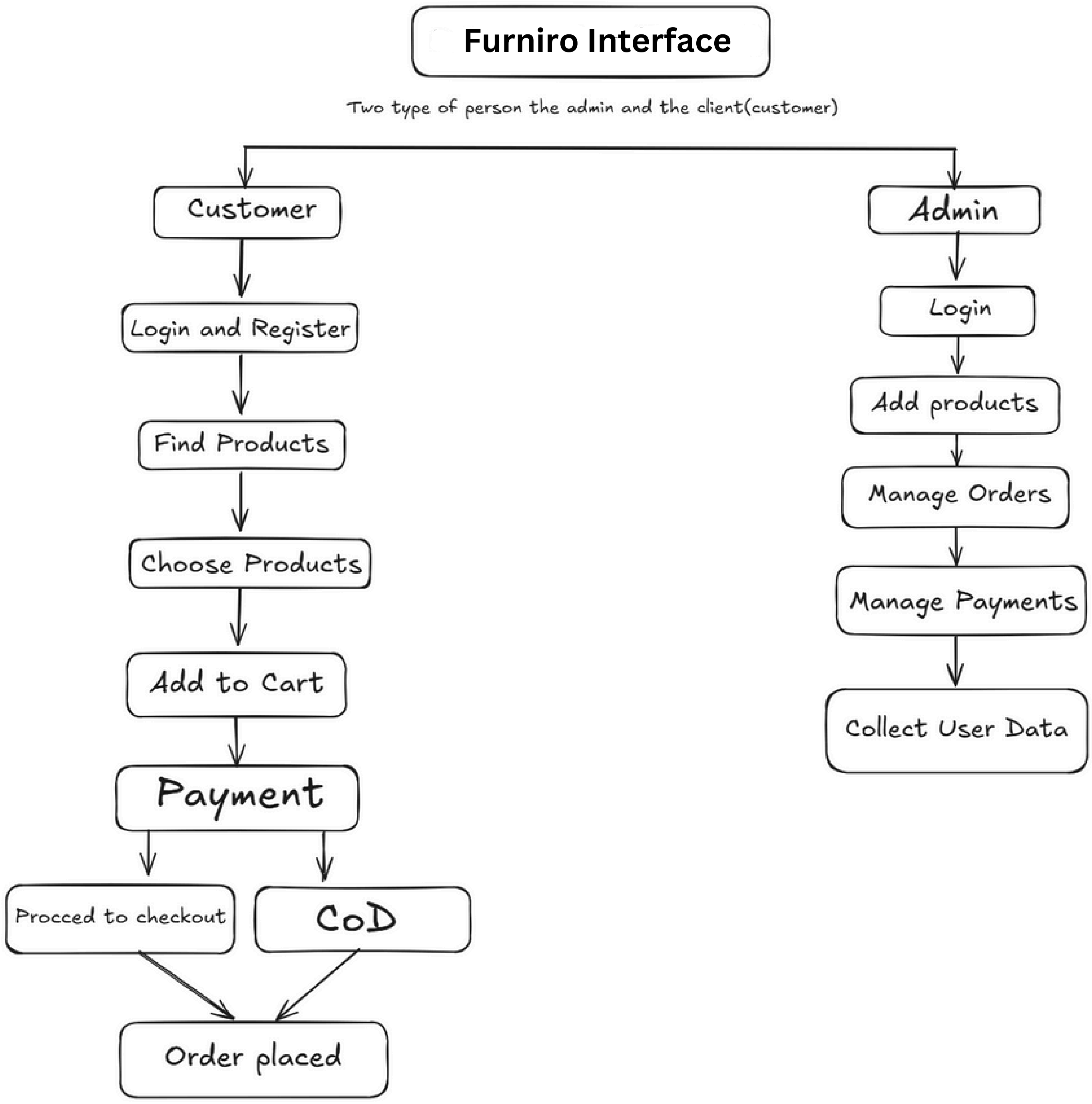
- Zendesk API: Customer support and ticket management.
- LiveChat API: Real-time chat widget integration for customer support.
- Intercom API: Customer communication and support with automated chatbots.

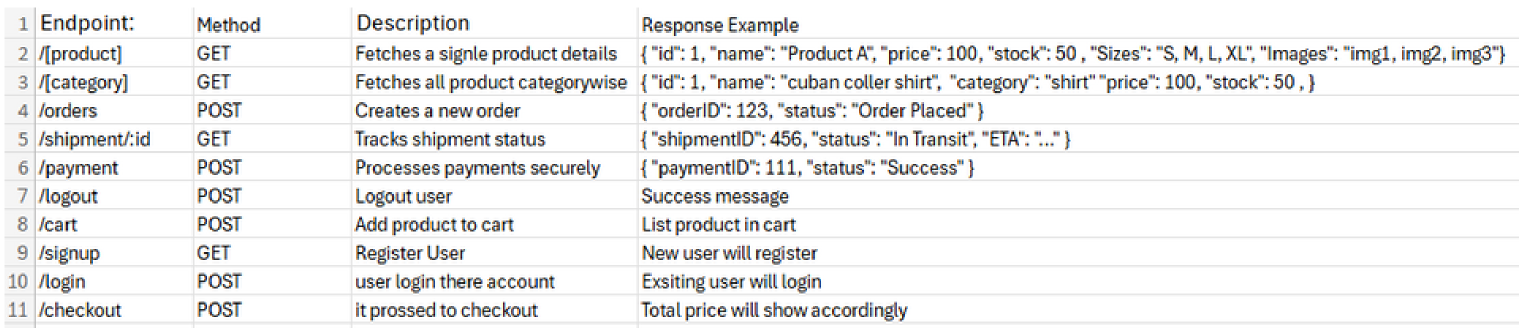
System Architecture of Furniro



- Frontend (Next.js): Responsive UI for browsing, cart, and checkout.
- Backend APIs: Manages data flow, business logic, and integrations.
- Sanity CMS: Stores and manages product, order, and user data.
- Third-Party APIs: Enables payments, shipment tracking, and notifications.
- Infrastructure (Vercel): Hosts the frontend with optimized CDNs.

Work Flow


Furniro Interface
Two type of person the admin and the client(customer)
Customer
Login and Register
Find Products
Choose Products
Add to Cart
Payment
Procced to checkout
CoD
Order placed
Admin
Login
Add products
Manage Orders
Manage Payments
Collect User Data



Endpoint:	Method	Description	Response Example
/[product]	GET	Fetches a signle product details	{ "id": 1, "name": "Product A", "price": 100, "stock": 50 , "Sizes": "S, M, L, XL", "Images": "img1, img2, img3"}
/[category]	GET	Fetches all product categorywise	{ "id": 1, "name": "cuban coller shirt", "category": "shirt" "price": 100, "stock": 50 , }
/orders	POST	Creates a new order	{ "orderID": 123, "status": "Order Placed" }
/shipment/:id	GET	Tracks shipment status	{ "shipmentID": 456, "status": "In Transit", "ETA": "..." }
/payment	POST	Processes payments securely	{ "paymentID": 111, "status": "Success" }
/logout	POST	Logout user	Success message
/cart	POST	Add product to cart	List product in cart
/signup	GET	Register User	New user will register
/login	POST	user login there account	Exsiting user will login
/checkout	POST	it prossed to checkout	Total price will show accordingly

Here’s the briefly explaination of all api endpoints:

/**orders**:
- Used to create a new order or fetch order details.
- Typically POST for creating an order and GET for fetching order history or specific order details.

/**[product]**:
- Used to fetch a specific product by its ID.
- Supports GET requests to retrieve product details like name, price, description, etc.

/[category]:
- Used to fetch products by category.
- Supports GET requests to retrieve a list of products in a specific category (e.g., electronics, fashion).

/**shipment/id**:
- Used to track shipment status using the order or shipment ID.
- Supports GET requests to retrieve the current status of the shipment, estimated delivery time, etc.

/**payment**:
- Used to process payments for orders.
- Typically POST for submitting payment details and completing the transaction.
- Returns success/failure based on the payment processing result.

/**logout**:
- Used to log the user out of the system.
- Typically POST or GET request to end the user session.

/**login**:
- Used to authenticate users and create a session.
- Typically POST with user credentials (username/email and password) to obtain a session token or authentication.

/**signup**:
- Used to register a new user in the system.
- Typically POST with user details like name, email, password, etc.

/**cart**:
- Used to add, update, or remove products from the cart.
- POST to add items, PUT to update quantity, DELETE to remove items from the cart.

/**checkout**:
- Used to initiate the checkout process.
- Typically POST to submit the cart details, shipping address, and payment information to complete the order.

Technical Roadmap:

Phase 1: Frontend Development:

Tasks:

- Design and develop responsive UI components for product listings, categories, and the shopping cart.
- Implement Tailwind CSS for styling.
- Integrate basic API calls to fetch products and categories from Sanity.

Milestones:

- Completion of product listing and category pages.
- Functional shopping cart UI.

Deliverables:

- A responsive frontend UI with dummy data rendering from APIs.

Phase 2: Backend Integration:

Tasks:

- Develop API endpoints for fetching products (/products), managing orders (/orders), and tracking shipments (/shipment).
- Integrate Sanity CMS with the frontend to manage real data dynamically.

Milestones:

- API endpoints functional and tested locally.
- Sanity CMS fully integrated with the frontend.

Deliverables:

- API documentation for endpoints.
- End-to-end data flow from CMS to the frontend.

Phase 3: Third-party APIs:

Tasks:

- Integrate a payment gateway (e.g., Stripe, PayPal).
- Add shipment tracking via third-party APIs.
- Build user authentication and profile management.

Milestones:

- Secure payment processing implemented.
- Shipment tracking operational.

Deliverables:

- Payment gateway integration.
- Functional user account management system.

Phase 4: Testing and Optimization

Tasks:

- Conduct end-to-end testing for all workflows, including product browsing, checkout, and order tracking.
- Optimize performance for fast loading and smooth UX.
- Resolve cross-browser and device compatibility issues.

Milestones:

- Completion of testing cycles.
- Performance benchmarks achieved.

Deliverables:

- Bug-free and optimized application ready for deployment.

Phase 5: Deployment and Maintenance

Tasks:

- Deploy the application to Vercel for hosting.
- Set up monitoring and error reporting tools.
- Plan for regular updates and content additions.

Milestones:

- Successful deployment of Robiz.
- Monitoring tools active.

Deliverables:

- Live, fully functional e-commerce platform.
- Post-deployment support and roadmap for future updates.

Sanity Schema

```
export const home = defineType({
 name: 'home_furniture',
 title: 'Home',
 type: 'document',
 fields: [
  defineField({
   name: 'title',
   title: 'Product Title',
   type: 'string',
   validation: (Rule) => Rule.required().min(3).max(150),
  }),
  defineField({
   name: 'slug',
   title: 'Slug',
   type: 'slug',
   options: {
    source: (doc: SanityDocument) => {
     const randomSuffix = Math.floor(Math.random() * 1000000000);
     if (doc.title) {
      return `${doc.title.replace(/\s+/g, '-').toLowerCase()}-${randomSuffix}`;
     }
     return randomSuffix.toString();
    },
    maxLength: 96,
   },
   validation: (Rule) => Rule.required(),
  }),
  defineField({
   name: 'sku',
   title: 'SKU',
   type: 'string',
   readOnly: true,
   initialValue: ({ document }) => {
    const categoryCode = document?.category?.slice(0, 2).toUpperCase() || 'F';
    const randomId = Math.floor(100000000 + Math.random() * 900000000);
    return `${categoryCode}${randomId}`;
   },
   validation: (Rule) => Rule.required(),
  }),
  defineField({
   name: 'overview',
   title: 'Product Overview',
   type: 'string',
  }),
  defineField({
   name: 'productdetails',
   type: 'array',
   title: 'Product Details',
   of: [
    {
     type: 'block',
    },
    {
     type: 'image',
     fields: [
      {
       type: 'text',
       name: 'alt',
       title: 'Alternative text',
      },
     ],
    },
   ],
  }),
defineField({
  name: 'additionalInformation',
  type: 'array',
  title: 'Additional Information',
  of: [
   {
    type: 'block',
   },
   {
    type: 'image',
    fields: [
     {
      type: 'text',
      name: 'alt',
      title: 'Alternative text',
     },
    ],
   },
  ],
 }),
 defineField({
  name: 'productReviews',
  type: 'object',
  title: 'Product Reviews',
  fields: [
   {
    name: 'averageRating',
    type: 'number',
    title: 'Average Rating',
    validation: (Rule) => Rule.min(0).max(5),
   },
   {
    name: 'totalReviews',
    type: 'number',
    title: 'Total Reviews',
   },
   {
    name: 'reviews',
    type: 'array',
    title: 'Individual Reviews',
    of: [
     {
      type: 'object',
      fields: [
       {
        name: 'reviewername',
        type: 'string',
        title: 'Reviewer Name',

       },
       {
        name: 'rating',
        type: 'number',
        title: 'Rating',
        validation: (Rule) => Rule.min(1).max(5),
       },
       {
        name: 'comment',
        type: 'text',
        title: 'Comment',
       },
      ],
     },
    ],
   },
  ],
 }),
```