GoSewa: Module-Wise Frontend-Backend Co-Alignment Guide

Full-Stack Development Team

January 8, 2026

1 Module 1: Authentication System

1.1 Shared Types

```typescript
// shared/types/auth.ts
export interface LoginRequest {
  email: string;
  password: string;
}

export interface LoginResponse {
  success: boolean;
  token: string;
  refreshToken: string;
  user: {
    id: string;
    email: string;
    role: 'consumer' | 'vendor' | 'admin';
    name: string;
  };
}

export interface RegisterRequest {
  email: string;
  password: string;
  name: string;
  phone?: string;
  role: 'consumer' | 'vendor';
  vendorDetails?: {
    businessName: string;
    gstNumber?: string;
  };
}
```

1.2 Backend Prompt

Backend Implementation Prompt

Create Authentication API with: 1. **File:** backend/src/routes/auth.routes.ts 2. **Endpoints:** - POST /api/auth/register - POST /api/auth/login - POST /api/auth/logout - POST /api/auth/refresh - GET /api/auth/me 3. **Middleware:** auth.middleware.ts for JWT verification 4. **Validation:** Use Zod with shared types 5. **Password:** bcrypt hashing, salt rounds: 10 6. **Token:** JWT with 15min expiry, refresh token 7 days 7. **Response:** Follow ApiResponse¡T¿ pattern

1.3 Frontend Prompt

Frontend Implementation Prompt

Create Authentication Components: 1. **Context:** auth.context.tsx with React Context + localStorage 2. **Hook:** useAuth() hook for easy auth access 3. **Components:** - LoginForm.tsx (with Formik/Yup validation) - RegisterForm.tsx (with role selection) - ProtectedRoute.tsx (HOC for route protection) 4. **Services:** auth.service.ts with axios interceptors 5. **Persistence:** Store tokens in localStorage with encryption 6. **Auto Refresh:** Implement token refresh on 401 7. **UI States:** Loading, error, success states for all forms

2 Module 2: Vendor Management

2.1 Shared Types

```
// shared/types/vendor.ts
export interface Vendor {
  id: string;
  userId: string;
  businessName: string;
  description?: string;
  gstNumber?: string;
  address: {
    street: string;
    city: string;
    state: string;
    pincode: string;
    coordinates?: {
      lat: number;
      lng: number;
    };
  };
  isVerified: boolean;
  isActive: boolean;
  rating?: number;
  totalSales?: number;
  createdAt: Date;
}
```

```
export interface VendorCreateRequest {
  userId: string;
  businessName: string;
  description?: string;
  gstNumber?: string;
  address: Vendor['address'];
}

export interface VendorUpdateRequest {
  businessName?: string;
  description?: string;
  isActive?: boolean;
}
```

2.2 Backend Prompt

Backend Vendor API Prompt

Create Vendor Management API: 1. **File:** backend/src/routes/vendor.routes.ts 2. **Endpoints:** - GET /api/vendors (admin: all, vendor: own) - GET /api/vendors/:id (public details) - POST /api/vendors (vendor registration) - PUT /api/vendors/:id (update vendor) - PATCH /api/vendors/:id/verify (admin: verify vendor) - GET /api/vendors/nearby?lat=lng=radius= (geolocation) 3. **Middleware:** Check vendor ownership/admin rights 4. **Geolocation:** Use PostGIS or calculate distance 5. **File Upload:** For vendor documents/KYC 6. **Validation:** GST number format, address validation

2.3 Frontend Prompt

Frontend Vendor Components Prompt

Create Vendor Components: 1. **Vendor Dashboard:** - VendorDashboard.tsx (main layout) - VendorStats.tsx (sales, orders, ratings) - ProductManagement.tsx (CRUD for products) 2. **Vendor Registration:** - Multi-step registration wizard - File upload for documents - Address with map picker 3. **Admin Vendor Management:** - VendorList.tsx (with filters/search) - VendorVerification.tsx (approve/reject) - VendorAnalytics.tsx (sales reports) 4. **Geolocation:** Show vendors on map 5. **Real-time:** WebSocket for order notifications

3 Module 3: Product Management

3.1 Shared Types

```
// shared/types/product.ts
export interface Product {
  id: string;
  vendorId: string;
  name: string;
```

```
  description: string;
  price: number;
  category: 'dairy' | 'organic' | 'handicrafts' | 'groceries';
  subCategory?: string;
  images: string[];
  stockQuantity: number;
  minOrderQuantity: number;
  maxOrderQuantity: number;
  isAvailable: boolean;
  tags: string[];
  attributes: Record<string, any>;
  createdAt: Date;
  updatedAt: Date;
}

export interface ProductCreateRequest {
  name: string;
  description: string;
  price: number;
  category: Product['category'];
  stockQuantity: number;
  images: File[];
}

export interface ProductFilter {
  category?: string;
  minPrice?: number;
  maxPrice?: number;
  vendorId?: string;
  search?: string;
  inStock?: boolean;
  page?: number;
  limit?: number;
  sortBy?: 'price' | 'createdAt' | 'rating';
  sortOrder?: 'asc' | 'desc';
}
```

3.2 Backend Prompt

Backend Product API Prompt

Create Product Management API: 1. **File:** backend/src/routes/product.routes.ts 2. **Endpoints:** - GET /api/products (public listing with filters) - GET /api/products/:id (product details) - POST /api/products (vendor: create product) - PUT /api/products/:id (vendor: update product) - DELETE /api/products/:id (vendor: soft delete) - GET /api/products/vendor/:vendorId (vendor's products) - POST /api/products/:id/images (upload more images) 3. **Image Upload:** Use Multer + Cloudinary/S3 4. **Search:** Full-text search on name/description 5. **Validation:** Price ¿ 0, stock ¿= 0 6. **Caching:** Redis cache for product listings 7. **Pagination:** Offset/limit with total count

3.3 Frontend Prompt

Frontend Product Components Prompt

Create Product Components: 1. **Product Listing:** - ProductGrid.tsx (responsive grid) - ProductFilters.tsx (category, price range) - ProductSearch.tsx (with autocomplete) - InfiniteScroll.tsx (pagination) 2. **Product Detail:** - ProductDetail.tsx (images, description, reviews) - ProductGallery.tsx (image carousel) - AddToCart.tsx (quantity selector) 3. **Vendor Product Management:** - ProductForm.tsx (create/edit product) - ProductImagesUpload.tsx (drag drop) - InventoryManagement.tsx (stock updates) 4. **Optimization:** - Image lazy loading - Product card skeleton - Offline product viewing

4 Module 4: Order Management

4.1 Shared Types

```
// shared/types/order.ts
export interface OrderItem {
  productId: string;
  quantity: number;
  price: number;
  total: number;
}

export interface Order {
  id: string;
  orderNumber: string;
  consumerId: string;
  vendorId: string;
  items: OrderItem[];
  subtotal: number;
  tax: number;
  shipping: number;
  total: number;
  status: 'pending' | 'confirmed' | 'processing' | 'shipped' | 'delivered' | 'cancelled'
  paymentStatus: 'pending' | 'paid' | 'failed' | 'refunded';
  shippingAddress: Address;
  billingAddress?: Address;
  notes?: string;
  createdAt: Date;
  updatedAt: Date;
}

export interface CreateOrderRequest {
  items: Array<{
    productId: string;
    quantity: number;
  }>;
  shippingAddress: Address;
  notes?: string;
```

```
}

export interface OrderStatusUpdate {
  status: Order['status'];
  trackingNumber?: string;
  notes?: string;
}
```

4.2 Backend Prompt

Backend Order API Prompt

Create Order Management API: 1. **File:** backend/src/routes/order.routes.ts 2. **Endpoints:** - POST /api/orders (create order) - GET /api/orders (user's orders) - GET /api/orders/:id (order details) - GET /api/orders/vendor/:vendorId (vendor's orders) - PUT /api/orders/:id/status (update status) - POST /api/orders/:id/cancel (cancel order) - GET /api/orders/:id/invoice (generate invoice) 3. **Business Logic:** - Validate stock before order - Calculate totals with tax - Generate order number (GOSW-YYYYMMDD-XXXX) - Send email notifications 4. **Validation:** Check product availability 5. **Transactions:** Use Prisma transactions for stock update

4.3 Frontend Prompt

Frontend Order Components Prompt

Create Order Components: 1. **Shopping Cart:** - CartProvider.tsx (context for cart) - CartSidebar.tsx (slide-out cart) - CartItem.tsx (individual items) 2. **Checkout Flow:** - CheckoutWizard.tsx (multi-step) - AddressForm.tsx (address management) - PaymentMethod.tsx (Razorpay integration) - OrderSummary.tsx (price breakdown) 3. **Order Tracking:** - OrderList.tsx (user's orders) - OrderDetail.tsx (timeline view) - OrderStatusTracker.tsx (progress bar) 4. **Vendor Order Management:** - VendorOrders.tsx (table with filters) - OrderStatusUpdater.tsx (status changes) - BulkActions.tsx (process multiple orders)

5 Module 5: Payment System

5.1 Shared Types

```
// shared/types/payment.ts
export interface PaymentRequest {
  orderId: string;
  amount: number;
  currency: 'INR';
  method: 'card' | 'netbanking' | 'upi' | 'wallet';
  customer: {
    email: string;
    phone: string;
    name: string;
  };
```

```
}

export interface PaymentResponse {
  paymentId: string;
  orderId: string;
  amount: number;
  currency: string;
  status: 'created' | 'authorized' | 'captured' | 'failed' | 'refunded';
  razorpayOrderId?: string;
  razorpayPaymentId?: string;
  razorpaySignature?: string;
  createdAt: Date;
}

export interface PaymentWebhook {
  event: string;
  payload: any;
  signature: string;
}
```

5.2 Backend Prompt

Backend Payment API Prompt

Create Payment Integration: 1. **File:** backend/src/routes/payment.routes.ts 2. **Endpoints:** - POST /api/payments/create (create payment intent) - POST /api/payments/verify (verify payment) - POST /api/payments/webhook (Razorpay webhook) - POST /api/payments/:id/refund (initiate refund) - GET /api/payments/order/:orderId (payment status) 3. **Integration:** - Razorpay SDK integration - Webhook signature verification - Idempotency keys for payments - Payment status sync with orders 4. **Security:** Never expose secret key 5. **Logging:** Complete payment audit trail

5.3 Frontend Prompt

Frontend Payment Components Prompt

Create Payment Components: 1. **Payment Integration:** - loadRazorpayScript.ts (dynamic script loading) - PaymentButton.tsx (initiate payment) - PaymentModal.tsx (Razorpay checkout) 2. **Payment Status:** - PaymentSuccess.tsx (success page) - PaymentFailed.tsx (retry options) - PaymentProcessing.tsx (loading state) 3. **Payment Methods:** - PaymentMethodSelector.tsx (cards, UPI, etc.) - SavedCards.tsx (card management) - UPIScanner.tsx (QR code scanner) 4. **Security:** - PCI compliance considerations - No sensitive data in frontend - Secure iframe for payment

6 Module 6: Admin Dashboard

6.1 Shared Types

```
// shared/types/admin.ts
```

```typescript
export interface DashboardStats {
  totalUsers: number;
  totalVendors: number;
  totalProducts: number;
  totalOrders: number;
  totalRevenue: number;
  pendingVerifications: number;
  recentOrders: Order[];
  topProducts: Array<{
    productId: string;
    name: string;
    sales: number;
  }>;
  revenueByDay: Array<{
    date: string;
    revenue: number;
  }>;
}

export interface AdminReportRequest {
  startDate: Date;
  endDate: Date;
  reportType: 'sales' | 'users' | 'inventory' | 'revenue';
  format?: 'json' | 'csv' | 'pdf';
}

export interface SystemSettings {
  platformCommission: number;
  taxRate: number;
  minOrderAmount: number;
  deliveryRadius: number;
  maintenanceMode: boolean;
}
```

6.2 Backend Prompt

Backend Admin API Prompt

Create Admin Dashboard API: 1. **File:** backend/src/routes/admin.routes.ts 2. **Endpoints:** - GET /api/admin/dashboard/stats (dashboard data) - GET /api/admin/users (user management) - GET /api/admin/reports (generate reports) - GET /api/admin/logs (system logs) - PUT /api/admin/settings (update settings) - POST /api/admin/broadcast (send notifications) 3. **Analytics:** - Aggregate queries for stats - Daily revenue calculation - User growth metrics 4. **Security:** Admin role middleware 5. **Caching:** Dashboard stats cache for 5 minutes

6.3 Frontend Prompt

Frontend Admin Components Prompt

Create Admin Dashboard: 1. **Dashboard Overview:** - StatsCards.tsx (KPI cards) - RevenueChart.tsx (line chart) - RecentActivity.tsx (activity feed) 2. **Management Pages:** - UserManagement.tsx (CRUD operations) - VendorApprovals.tsx (bulk actions) - SystemSettings.tsx (form with validation) 3. **Reports:** - ReportGenerator.tsx (date range, filters) - DataTable.tsx (export to CSV/Excel) - Charts.tsx (Recharts/Chart.js) 4. **Real-time Updates:** - LiveStats.tsx (WebSocket updates) - NotificationCenter.tsx (admin alerts)

7 Integration Testing Prompts

7.1 End-to-End Test Prompt

E2E Test Implementation Prompt

Create Integration Tests: 1. **Test File:** tests/integration/order-flow.test.ts 2. **Scenario:** Complete order flow 3. **Steps:** - User registers/login - Browses products - Adds to cart - Checks out with address - Makes payment - Vendor processes order - User receives order 4. **Tools:** Cypress/Playwright 5. **Assertions:** Verify database state matches UI 6. **Mocking:** Mock payment gateway for testing

7.2 API Contract Test Prompt

API Contract Test Prompt

Create Contract Tests: 1. **Tool:** Use Pact or OpenAPI validator 2. **Consumer:** Frontend defines expected API 3. **Provider:** Backend verifies it meets contract 4. **Test Cases:** - Request/response schemas - HTTP status codes - Error responses - Headers and authentication 5. **CI/CD:** Run on every PR

8 Deployment Coordination

8.1 Environment Variables Alignment

```
# .env.example (Shared)
NEXT_PUBLIC_API_URL=http://localhost:5000/api
NEXT_PUBLIC_WS_URL=ws://localhost:5000
NEXT_PUBLIC_RAZORPAY_KEY=rzp_test_xxx

# Backend .env
DATABASE_URL=postgresql://...
JWT_SECRET=...
RAZORPAY_SECRET=...
AWS_ACCESS_KEY=...
```

```
# Frontend .env.local
NEXT_PUBLIC_API_URL=https://api.gosewa.com/api
NEXT_PUBLIC_WS_URL=wss://api.gosewa.com
```

8.2 Docker Compose Setup

```
version: '3.8'
services:
  postgres:
    image: postgres:15
    environment:
      POSTGRES_DB: gosewa
      POSTGRES_USER: admin
      POSTGRES_PASSWORD: password
    volumes:
      - postgres_data:/var/lib/postgresql/data

  redis:
    image: redis:7-alpine

  backend:
    build: ./backend
    ports:
      - "5000:5000"
    environment:
      DATABASE_URL: postgresql://admin:password@postgres:5432/gosewa
      REDIS_URL: redis://redis:6379
    depends_on:
      - postgres
      - redis

  frontend:
    build: ./frontend
    ports:
      - "3000:3000"
    environment:
      NEXT_PUBLIC_API_URL: http://backend:5000/api
    depends_on:
      - backend
```

9 Development Workflow

9.1 Git Branch Strategy

```
# Feature branches for each module
git checkout -b feat/auth-system
git checkout -b feat/vendor-management
git checkout -b feat/product-catalog

# Commit convention
```

```
git commit -m "feat(auth): implement login API"
git commit -m "feat(frontend): add login form"
git commit -m "test: add auth integration tests"

# PR Template includes:
- [ ] Backend changes complete
- [ ] Frontend changes complete
- [ ] Integration tests passing
- [ ] Shared types updated
```

9.2 Code Review Checklist

- **Backend Reviewer:** - API follows shared types - Proper error handling - Security considerations - Database queries optimized
- **Frontend Reviewer:** - Uses shared types - Proper error states - Responsive design - Accessibility considerations
- **Integration Reviewer:** - End-to-end flow works - API contracts maintained - Performance acceptable - Mobile responsiveness

10 Monitoring & Debugging

10.1 Shared Logging

```
// shared/logger.ts
export const createLogger = (module: string) => ({
  info: (message: string, data?: any) =>
    console.log(`[${module}] ${message}`, data),
  error: (message: string, error?: any) =>
    console.error(`[${module}] ERROR: ${message}`, error),
  warn: (message: string) =>
    console.warn(`[${module}] WARN: ${message}`),
});

// Usage in both frontend and backend
const logger = createLogger('auth');
logger.info('User logged in', { userId: '123' });
```

10.2 Error Tracking

```
// shared/errorHandler.ts
export class AppError extends Error {
  constructor(
    public message: string,
    public statusCode: number = 500,
    public code?: string
  ) {
    super(message);
  }
}
```

```
// Backend usage
throw new AppError('Product not found', 404, 'PRODUCT_NOT_FOUND');

// Frontend usage
if (error instanceof AppError) {
  showToast(error.message);
}
```