Phase 1: Backend Development Setup Plan

🎯 Setup Strategy

We'll create a backend that perfectly matches your Flutter app's API expectations, using the same clean architecture principles.

📁 Project Structure

```
backend/
├── src/
│   ├── config/          # Database, JWT, environment configs
│   ├── controllers/     # Route handlers (API endpoints)
│   ├── middleware/      # Auth, validation, error handling
│   ├── models/          # Database models (Sequelize/Prisma)
│   ├── routes/          # API route definitions
│   ├── services/        # Business logic layer
│   ├── utils/           # Helper functions
│   └── validators/      # Request validation schemas
├── migrations/          # Database migrations
├── seeders/            # Sample data
├── tests/              # API tests
├── docker-compose.yml   # PostgreSQL + Redis setup
├── Dockerfile          # Backend containerization
├── package.json
└── README.md
```

🛠️ Technology Stack

Core Framework

- **Node.js** + **Express.js** - Fast, lightweight API
- **TypeScript** - Type safety and better development experience

Database & ORM

- **PostgreSQL** - Robust relational database
- **Prisma** - Modern ORM with excellent TypeScript support
- **Redis** - Caching and session storage

Authentication & Security

- **JWT** - Token-based authentication
- **bcrypt** - Password hashing

- **helmet** - Security headers
- **cors** - Cross-origin resource sharing

Validation & Documentation

- **Joi** or **Zod** - Request validation
- **Swagger** - API documentation
- **Morgan** - HTTP request logging

Real-time Features

- **Socket.io** - WebSocket for real-time order tracking

🚀 Setup Steps

Step 1: Initialize Backend Project

```bash
# Create backend directory
mkdir backend && cd backend

# Initialize Node.js project
npm init -y

# Install core dependencies
npm install express cors helmet morgan compression
npm install jsonwebtoken bcryptjs joi swagger-ui-express
npm install prisma @prisma/client socket.io
npm install redis ioredis

# Install dev dependencies
npm install -D typescript @types/node @types/express
npm install -D @types/cors @types/helmet @types/morgan
npm install -D @types/jsonwebtoken @types/bcryptjs
npm install -D nodemon ts-node dotenv
npm install -D jest @types/jest supertest @types/supertest
```

Step 2: Database Setup

bash

```bash
# Initialize Prisma
npx prisma init

# This creates:
# - prisma/schema.prisma
# - .env file
```

Step 3: Docker Setup for Development

yaml

```yaml
# docker-compose.yml for local development
version: '3.8'
services:
  postgres:
    image: postgres:15
    environment:
      POSTGRES_DB: delivery_system
      POSTGRES_USER: admin
      POSTGRES_PASSWORD: password
    ports:
      - "5432:5432"
    volumes:
      - postgres_data:/var/lib/postgresql/data

  redis:
    image: redis:7-alpine
    ports:
      - "6379:6379"
    volumes:
      - redis_data:/data

volumes:
  postgres_data:
  redis_data:
```

📊 Database Schema Design

Core Tables

1. **users** - All user types (customers, vendors, delivery, admin)
2. **shops** - Vendor shops/restaurants
3. **products** - Items sold by shops

4. **orders** - Customer orders
5. **order_items** - Products in each order
6. **addresses** - User delivery addresses
7. **payments** - Payment records
8. **reviews** - User reviews for shops/products

Key Relationships

- Users → Shops (1:many for vendors)
- Shops → Products (1:many)
- Users → Orders (1:many)
- Orders → OrderItems (1:many)
- Products → OrderItems (1:many)

🔧 API Endpoints Structure

Authentication Routes

```
POST /api/auth/register
POST /api/auth/login
POST /api/auth/logout
POST /api/auth/refresh
POST /api/auth/forgot-password
POST /api/auth/reset-password
GET  /api/auth/me
```

Shop Routes

```
GET    /api/shops              # Get shops with filters
GET    /api/shops/featured     # Featured shops
GET    /api/shops/nearby       # Nearby shops
GET    /api/shops/:id          # Get shop details
GET    /api/shops/:id/products # Get shop products
GET    /api/shops/:id/categories # Get product categories
```

Product Routes

```
GET    /api/products/:id       # Get product details
```

Order Routes

```
POST   /api/orders             # Place order
GET    /api/orders             # Get user orders
GET    /api/orders/:id         # Get order details
PATCH  /api/orders/:id/cancel  # Cancel order
PATCH  /api/orders/:id/tip     # Update tip
```

Vendor Routes

```
GET    /api/vendor/orders      # Get shop orders
PATCH  /api/vendor/orders/:id/status # Update order status
GET    /api/vendor/shops/:id   # Get vendor shop
PUT    /api/vendor/shops/:id   # Update shop
POST   /api/vendor/products    # Create product
PUT    /api/vendor/products/:id # Update product
```

Delivery Routes

```
GET    /api/delivery/orders    # Get assigned orders
PATCH  /api/delivery/orders/:id/location # Update location
PATCH  /api/delivery/orders/:id/delivered # Mark delivered
```

🔐 Security Implementation

JWT Strategy

- Access tokens (15 minutes)
- Refresh tokens (7 days)
- Role-based access control

Password Security

- bcrypt with salt rounds: 12
- Password strength validation

API Security

- Rate limiting
- Input sanitization
- SQL injection prevention
- CORS configuration

📈 Development Workflow

Phase 1A: Core Setup (Week 1)

1. Project initialization
2. Database schema design
3. Basic Express server
4. Authentication middleware

Phase 1B: Auth System (Week 2)

1. User registration/login
2. JWT implementation
3. Password reset flow
4. Role-based middleware

Phase 1C: Core APIs (Weeks 3-4)

1. Shop management APIs
2. Product management APIs
3. Order system APIs
4. Basic validation

Phase 1D: Advanced Features (Weeks 5-6)

1. Real-time order tracking
2. File upload for images
3. Search and filtering
4. API documentation

🧪 Testing Strategy

Test Types

- **Unit Tests** - Individual functions
- **Integration Tests** - API endpoints
- **E2E Tests** - Complete workflows

Test Coverage

- Authentication flows
- CRUD operations
- Business logic
- Error handling

📝 Documentation

API Documentation

- Swagger/OpenAPI specification
- Request/response examples
- Error code explanations

Development Docs

- Setup instructions
- Environment configuration
- Database migrations
- Deployment guide

🚀 Next Steps

1. **Choose your preferred approach:**
   - Start with basic Express setup
   - Use a starter template
   - Follow step-by-step creation
2. **Database preference:**
   - Prisma (recommended) - Modern, type-safe
   - Sequelize - Mature, feature-rich
   - Raw SQL with pg library
3. **Development environment:**
   - Local PostgreSQL installation
   - Docker-based development
   - Cloud database (development)

Would you like me to:

1. **Create the complete backend project structure** with all files?
2. **Start with just the basic setup** and build incrementally?
3. **Focus on the database schema** first?