Technical Documentation: Local Business Support Platform

1. System Overview

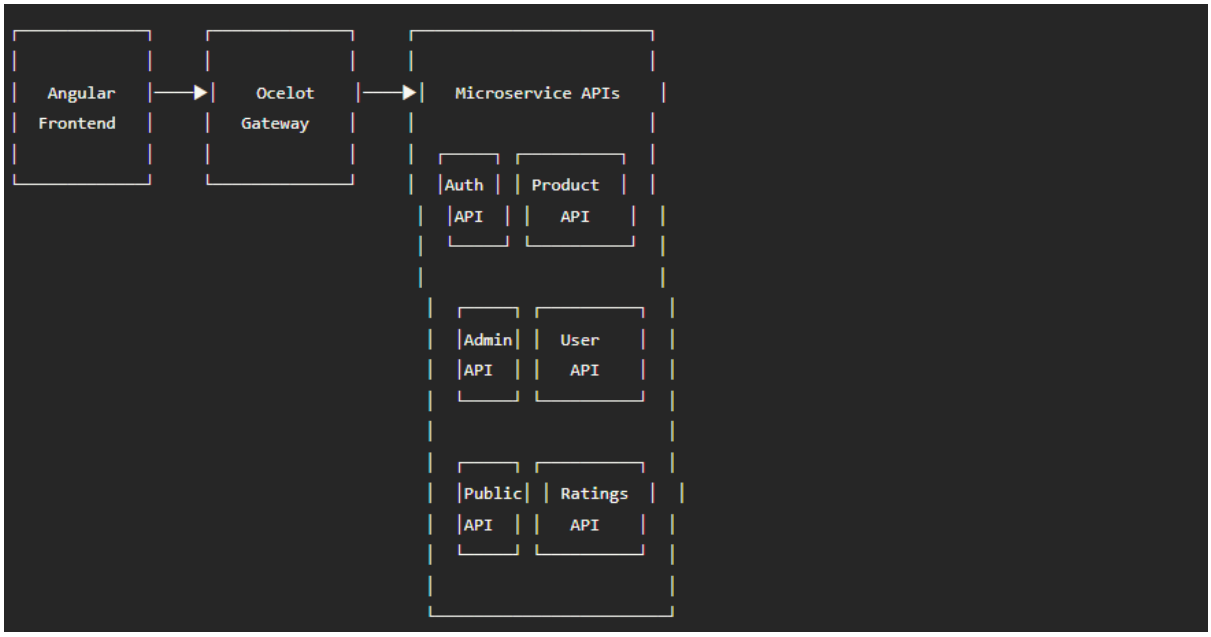
This document provides technical specifications for the Local Business Support Platform, a .NET-based application designed to connect users with local businesses, facilitate transactions, and promote community economic development.

2. Architecture

The system implements a microservices architecture with the following components:

- **Backend**: .NET Core-based RESTful APIs
- **API Gateway**: Ocelot
- **Authentication**: JWT-based token system
- **Frontend**: Angular SPA (Single Page Application)

2.1 Architecture Diagram



3. Backend Services

3.1 **AuthAPI**

Purpose: Manages authentication, authorization, and user identity.

Key Endpoints:

- POST /api/auth/register - Register new users
- POST /api/auth/login - Authenticate users and issue JWT
- POST /api/auth/refresh-token - Refresh expired JWTs
- POST /api/auth/change-password - Update user credentials
- GET /api/auth/validate - Validate token

Technologies:

- ASP.NET Core Identity
- Entity Framework Core
- JWT implementation using Microsoft.AspNetCore.Authentication.JwtBearer

3.2 ProductAPI

Purpose: Manages local business products and inventory.

Key Endpoints:

- GET /api/products - List all products
- GET /api/products/{id} - Get product details
- POST /api/products - Add new product (requires business owner permissions)
- PUT /api/products/{id} - Update product
- DELETE /api/products/{id} - Remove product
- GET /api/products/business/{businessId} - Get products for a specific business

3.3 AdminAPI

Purpose: Provides platform administration capabilities.

Key Endpoints:

- GET /api/admin/users - Manage users
- GET /api/admin/businesses - Approve and manage business listings
- POST /api/admin/reports - Generate platform reports
- PUT /api/admin/settings - Update system settings
- POST /api/admin/moderate - Content moderation endpoints

3.4 UserAPI

Purpose: Manages user profiles, preferences, and transactions.

Key Endpoints:

- GET /api/users/profile - Get current user profile
- PUT /api/users/profile - Update profile
- GET /api/users/favorites - Get favorite businesses
- POST /api/users/favorites/{businessId} - Add business to favorites
- GET /api/users/transactions - View purchase history

3.5 PublicAPI

Purpose: Provides public-facing data about local businesses.

Key Endpoints:

- GET /api/public/businesses - List businesses with filtering
- GET /api/public/businesses/{id} - Get business details
- GET /api/public/categories - Get business categories
- GET /api/public/search - Search functionality
- GET /api/public/trending - Get trending businesses

3.6 RatingsAPI

Purpose: Manages reviews and ratings for local businesses.

-
- GET /api/ratings/business/{businessId} - Get ratings for a business
- POST /api/ratings - Submit a new rating
- PUT /api/ratings/{id} - Update existing rating
- DELETE /api/ratings/{id} - Remove rating
- GET /api/ratings/user - Get ratings by current user

4. Authentication System

The platform uses JWT (JSON Web Tokens) for authentication across all services.

4.1 JWT Implementation

Token Structure: Standard JWT with header, payload, and signature

Claims:

1. User ID
2. Username
3. Roles (Admin, Business, Customer)
4. Expiration time

Token Lifetime: Access tokens expire after 60 minutes; refresh tokens after 7 days

Storage: Client stores tokens in browser local Storage with appropriate security measures

4.2 Authentication Flow

1. User logs in via AuthAPI

2. AuthAPI validates credentials and issues JWT

3. Client includes JWT in Authorization header for all subsequent requests

4. API Gateway (Ocelot) validates token before forwarding requests to microservices

5. Individual services perform specific authorization checks based on user claims

5. API Gateway

Ocelot is implemented as an API gateway to provide:

5.1 Configuration

```
{
  "Routes": [
    {
      "DownstreamPathTemplate": "/api/auth/{everything}",
      "DownstreamScheme": "https",
      "DownstreamHostAndPorts": [
        {
          "Host": "auth-api",
          "Port": 443
        }
      ],
      "UpstreamPathTemplate": "/api/auth/{everything}",
      "UpstreamHttpMethod": [ "GET", "POST", "PUT" ]
    },

    // Similar configurations for other APIs
  ],

  "GlobalConfiguration": {
    "BaseUrl": "https://api.localbusinesssupport.com"
  }
}
```

5.2 Gateway Features

a. Routing: Directs requests to appropriate microservices
b. Authentication: Validates JWTs before passing requests
c. Rate Limiting: Prevents abuse of the APIs
d. Request Aggregation: Combines responses from multiple services where needed
e. Logging: Centralized request logging
f. Caching: Response caching for improved performance

6. Frontend Architecture [Planned]

The frontend is built using Angular with the following structure:

6.1 Key Components

1. Core Module: Authentication services, HTTP interceptors, guards
2. Shared Module: Common components, pipes, and directives
3. Feature Modules:

    -User Dashboard

    - Business Profiles

    - Product Catalog

    - Shopping Cart

    - Order Management

    - Admin Dashboard

6.2 State Management

- NgRx store for application state management
- Feature-based state organization
- Optimistic UI updates

6.3 API Communication

- Angular HttpClient with interceptors for:
  - JWT attachment
  - Error handling
  - Loading indicators
  - Retry logic

7. Development Guidelines

7.1 API Development Standards

- RESTful design principles
- Versioning via URL path (e.g., /api/v1/resource)
- Consistent response formats
- HTTP status codes usage
- Comprehensive Swagger documentation

7.2 Security Measures

- HTTPS for all communications
- JWT validation on every request
- Input validation and sanitization
- CORS configuration
- Rate limiting for public endpoints
- Data encryption for sensitive information

8. Deployment Architecture

The system is designed for containerized deployment using Docker and orchestrated with Kubernetes.

8.1 Environment Configuration

- Development
- Testing
- Staging
- Production

Each environment has isolated configurations for databases, third-party integrations, and security settings.

This documentation is maintained by team 8 and should be updated as architecture evolves.