Shopzy E-commerce Development Guide

Phase 1: Project Setup & Database

Step 1: Create Database

1. **Create SQL Server Database**

```sql
CREATE DATABASE ShopzyDB;
```

2. **Execute the database schema** (from the previous artifact)

Step 2: Setup .NET Core Web API Project

1. **Create the solution and projects**

```bash
# Create solution
dotnet new sln -n Shopzy

# Create Web API project
dotnet new webapi -n Shopzy.API
dotnet sln add Shopzy.API

# Create Business Logic project
dotnet new classlib -n Shopzy.Business
dotnet sln add Shopzy.Business

# Create Data Access project
dotnet new classlib -n Shopzy.Data
dotnet sln add Shopzy.Data

# Create Models project
dotnet new classlib -n Shopzy.Models
dotnet sln add Shopzy.Models
```

2. **Add project references**

```bash
```

```
cd Shopzy.API
dotnet add reference ../Shopzy.Business
dotnet add reference ../Shopzy.Models

cd ../Shopzy.Business
dotnet add reference ../Shopzy.Data
dotnet add reference ../Shopzy.Models

cd ../Shopzy.Data
dotnet add reference ../Shopzy.Models
```

3. **Install NuGet packages**

```bash
# In Shopzy.API
dotnet add package Microsoft.EntityFrameworkCore.SqlServer
dotnet add package Microsoft.EntityFrameworkCore.Tools
dotnet add package Microsoft.AspNetCore.Authentication.JwtBearer
dotnet add package Microsoft.AspNetCore.Identity.EntityFrameworkCore
dotnet add package AutoMapper.Extensions.Microsoft.DependencyInjection
dotnet add package Swashbuckle.AspNetCore
dotnet add package FluentValidation.AspNetCore

# In Shopzy.Data
dotnet add package Microsoft.EntityFrameworkCore.SqlServer
dotnet add package Microsoft.EntityFrameworkCore.Tools
dotnet add package Microsoft.AspNetCore.Identity.EntityFrameworkCore
```

Phase 2: Data Models & Entity Framework

Step 3: Create Entity Models

Create models in `Shopzy.Models`:

User.cs

```csharp
```

```
public class User
{
    public int Id { get; set; }
    public string Email { get; set; }
    public string PasswordHash { get; set; }
    public string FirstName { get; set; }
    public string LastName { get; set; }
    public string PhoneNumber { get; set; }
    public DateTime? DateOfBirth { get; set; }
    public bool IsEmailVerified { get; set; }
    public bool IsActive { get; set; } = true;
    public DateTime CreatedAt { get; set; }
    public DateTime UpdatedAt { get; set; }

    // Navigation properties
    public virtual ICollection<UserRole> UserRoles { get; set; }
    public virtual ICollection<UserAddress> Addresses { get; set; }
    public virtual ICollection<Order> Orders { get; set; }
    public virtual ICollection<ProductReview> Reviews { get; set; }
    public virtual ICollection<Wishlist> Wishlists { get; set; }
}
```

Product.cs

```
csharp
```

```
public class Product
{
    public int Id { get; set; }
    public string Name { get; set; }
    public string Description { get; set; }
    public string ShortDescription { get; set; }
    public string SKU { get; set; }
    public decimal Price { get; set; }
    public decimal? CompareAtPrice { get; set; }
    public decimal? CostPrice { get; set; }
    public decimal? Weight { get; set; }
    public int? CategoryId { get; set; }
    public int? BrandId { get; set; }
    public int StockQuantity { get; set; }
    public int LowStockThreshold { get; set; } = 5;
    public bool IsActive { get; set; } = true;
    public bool IsFeatured { get; set; }
    public string MetaTitle { get; set; }
    public string MetaDescription { get; set; }
    public DateTime CreatedAt { get; set; }
    public DateTime UpdatedAt { get; set; }

    // Navigation properties
    public virtual Category Category { get; set; }
    public virtual Brand Brand { get; set; }
    public virtual ICollection<ProductImage> Images { get; set; }
    public virtual ICollection<ProductVariant> Variants { get; set; }
    public virtual ICollection<ProductReview> Reviews { get; set; }
    public virtual ICollection<CartItem> CartItems { get; set; }
    public virtual ICollection<OrderItem> OrderItems { get; set; }
}
```

Step 4: Create DbContext

ShopzyDbContext.cs in `Shopzy.Data`:

```
csharp
```

```
public class ShopzyDbContext : DbContext
{
    public ShopzyDbContext(DbContextOptions<ShopzyDbContext> options) : base(options)
    {
    }

    public DbSet<User> Users { get; set; }
    public DbSet<Role> Roles { get; set; }
    public DbSet<UserRole> UserRoles { get; set; }
    public DbSet<UserAddress> UserAddresses { get; set; }
    public DbSet<Category> Categories { get; set; }
    public DbSet<Brand> Brands { get; set; }
    public DbSet<Product> Products { get; set; }
    public DbSet<ProductImage> ProductImages { get; set; }
    public DbSet<ProductAttribute> ProductAttributes { get; set; }
    public DbSet<ProductAttributeValue> ProductAttributeValues { get; set; }
    public DbSet<ProductVariant> ProductVariants { get; set; }
    public DbSet<ShoppingCart> ShoppingCarts { get; set; }
    public DbSet<CartItem> CartItems { get; set; }
    public DbSet<Order> Orders { get; set; }
    public DbSet<OrderItem> OrderItems { get; set; }
    public DbSet<Payment> Payments { get; set; }
    public DbSet<Coupon> Coupons { get; set; }
    public DbSet<ProductReview> ProductReviews { get; set; }
    public DbSet<Wishlist> Wishlists { get; set; }

    protected override void OnModelCreating(ModelBuilder modelBuilder)
    {
        base.OnModelCreating(modelBuilder);

        // Configure relationships and constraints
        modelBuilder.Entity<UserRole>()
            .HasKey(ur => new { ur.UserId, ur.RoleId });

        modelBuilder.Entity<ProductVariantAttribute>()
            .HasKey(pva => new { pva.ProductVariantId, pva.AttributeValueId });

        modelBuilder.Entity<Wishlist>()
            .HasKey(w => new { w.UserId, w.ProductId });

        // Configure decimal precision
        modelBuilder.Entity<Product>()
            .Property(p => p.Price)
            .HasColumnType("decimal(18,2)");

        modelBuilder.Entity<Order>()
```

```
            .Property(o => o.TotalAmount)
            .HasColumnType("decimal(18,2)");

        // Add more configurations as needed
    }
}
```

Phase 3: Repository Pattern & Business Logic

Step 5: Create Repository Interface and Implementation

IRepository.cs in `Shopzy.Data`:

```csharp
public interface IRepository<T> where T : class
{
    Task<T> GetByIdAsync(int id);
    Task<IEnumerable<T>> GetAllAsync();
    Task<IEnumerable<T>> FindAsync(Expression<Func<T, bool>> expression);
    Task AddAsync(T entity);
    void Update(T entity);
    void Remove(T entity);
    Task SaveChangesAsync();
}
```

Repository.cs:

```csharp
```

```
public class Repository<T> : IRepository<T> where T : class
{
    protected readonly ShopzyDbContext _context;
    protected readonly DbSet<T> _dbSet;

    public Repository(ShopzyDbContext context)
    {
        _context = context;
        _dbSet = context.Set<T>();
    }

    public async Task<T> GetByIdAsync(int id)
    {
        return await _dbSet.FindAsync(id);
    }

    public async Task<IEnumerable<T>> GetAllAsync()
    {
        return await _dbSet.ToListAsync();
    }

    public async Task<IEnumerable<T>> FindAsync(Expression<Func<T, bool>> expression)
    {
        return await _dbSet.Where(expression).ToListAsync();
    }

    public async Task AddAsync(T entity)
    {
        await _dbSet.AddAsync(entity);
    }

    public void Update(T entity)
    {
        _dbSet.Update(entity);
    }

    public void Remove(T entity)
    {
        _dbSet.Remove(entity);
    }

    public async Task SaveChangesAsync()
    {
        await _context.SaveChangesAsync();
    }
}
```

Step 6: Create Business Services

IProductService.cs in `Shopzy.Business`:

```csharp
public interface IProductService
{
    Task<Product> GetProductByIdAsync(int id);
    Task<IEnumerable<Product>> GetProductsAsync();
    Task<IEnumerable<Product>> GetFeaturedProductsAsync();
    Task<IEnumerable<Product>> GetProductsByCategoryAsync(int categoryId);
    Task<Product> CreateProductAsync(Product product);
    Task<Product> UpdateProductAsync(Product product);
    Task DeleteProductAsync(int id);
    Task<IEnumerable<Product>> SearchProductsAsync(string searchTerm);
}
```

ProductService.cs:

```csharp
```

```
public class ProductService : IProductService
{
    private readonly IRepository<Product> _productRepository;
    private readonly IMapper _mapper;

    public ProductService(IRepository<Product> productRepository, IMapper mapper)
    {
        _productRepository = productRepository;
        _mapper = mapper;
    }

    public async Task<Product> GetProductByIdAsync(int id)
    {
        return await _productRepository.GetByIdAsync(id);
    }

    public async Task<IEnumerable<Product>> GetProductsAsync()
    {
        return await _productRepository.FindAsync(p => p.IsActive);
    }

    public async Task<IEnumerable<Product>> GetFeaturedProductsAsync()
    {
        return await _productRepository.FindAsync(p => p.IsFeatured && p.IsActive);
    }

    public async Task<Product> CreateProductAsync(Product product)
    {
        product.CreatedAt = DateTime.UtcNow;
        product.UpdatedAt = DateTime.UtcNow;
        await _productRepository.AddAsync(product);
        await _productRepository.SaveChangesAsync();
        return product;
    }

    // Implement other methods...
}
```

Phase 4: Web API Controllers

Step 7: Create API Controllers

ProductsController.cs in `Shopzy.API/Controllers`:

```
csharp
```

```csharp
[ApiController]
[Route("api/[controller]")]
public class ProductsController : ControllerBase
{
    private readonly IProductService _productService;
    private readonly IMapper _mapper;

    public ProductsController(IProductService productService, IMapper mapper)
    {
        _productService = productService;
        _mapper = mapper;
    }

    [HttpGet]
    public async Task<ActionResult<IEnumerable<ProductDto>>> GetProducts()
    {
        var products = await _productService.GetProductsAsync();
        return Ok(_mapper.Map<IEnumerable<ProductDto>>(products));
    }

    [HttpGet("{id}")]
    public async Task<ActionResult<ProductDto>> GetProduct(int id)
    {
        var product = await _productService.GetProductByIdAsync(id);
        if (product == null)
            return NotFound();

        return Ok(_mapper.Map<ProductDto>(product));
    }

    [HttpPost]
    [Authorize(Roles = "Admin")]
    public async Task<ActionResult<ProductDto>> CreateProduct(CreateProductDto createProductDto)
    {
        var product = _mapper.Map<Product>(createProductDto);
        var createdProduct = await _productService.CreateProductAsync(product);
        var productDto = _mapper.Map<ProductDto>(createdProduct);

        return CreatedAtAction(nameof(GetProduct), new { id = productDto.Id }, productDto);
    }

    // Add more endpoints...
}
```

Step 8: Configure Services in Startup/Program.cs

```
csharp

public void ConfigureServices(IServiceCollection services)
{
    // Database
    services.AddDbContext<ShopzyDbContext>(options =>
        options.UseSqlServer(Configuration.GetConnectionString("DefaultConnection")));

    // Repositories
    services.AddScoped(typeof(IRepository<>), typeof(Repository<>));

    // Services
    services.AddScoped<IProductService, ProductService>();
    services.AddScoped<IUserService, UserService>();
    services.AddScoped<IOrderService, OrderService>();

    // AutoMapper
    services.AddAutoMapper(typeof(MappingProfile));

    // Authentication
    services.AddAuthentication(JwtBearerDefaults.AuthenticationScheme)
        .AddJwtBearer(options =>
        {
            options.TokenValidationParameters = new TokenValidationParameters
            {
                ValidateIssuer = true,
                ValidateAudience = true,
                ValidateLifetime = true,
                ValidateIssuerSigningKey = true,
                ValidIssuer = Configuration["Jwt:Issuer"],
                ValidAudience = Configuration["Jwt:Audience"],
                IssuerSigningKey = new SymmetricSecurityKey(Encoding.UTF8.GetBytes(Configuration["Jwt:Key"]))
            };
        });

    services.AddControllers();
    services.AddSwaggerGen();
}
```

Phase 5: React.js Frontend

Step 9: Setup React Application

```
bash
```

```
# Create React app
npx create-react-app shopzy-frontend
cd shopzy-frontend

# Install additional packages
npm install axios react-router-dom @reduxjs/toolkit react-redux
npm install @mui/material @emotion/react @emotion/styled
npm install @mui/icons-material
npm install react-hook-form @hookform/resolvers yup
```

Step 10: Setup Redux Store

store/store.js:

javascript

```javascript
import { configureStore } from '@reduxjs/toolkit'
import authSlice from './slices/authSlice'
import productSlice from './slices/productSlice'
import cartSlice from './slices/cartSlice'

export const store = configureStore({
  reducer: {
    auth: authSlice,
    products: productSlice,
    cart: cartSlice,
  },
})
```

store/slices/productSlice.js:

javascript

```
import { createSlice, createAsyncThunk } from '@reduxjs/toolkit'
import api from '../../services/api'

export const fetchProducts = createAsyncThunk(
  'products/fetchProducts',
  async () => {
    const response = await api.get('/products')
    return response.data
  }
)

const productSlice = createSlice({
  name: 'products',
  initialState: {
    items: [],
    loading: false,
    error: null,
  },
  reducers: {},
  extraReducers: (builder) => {
    builder
      .addCase(fetchProducts.pending, (state) => {
        state.loading = true
      })
      .addCase(fetchProducts.fulfilled, (state, action) => {
        state.loading = false
        state.items = action.payload
      })
      .addCase(fetchProducts.rejected, (state, action) => {
        state.loading = false
        state.error = action.error.message
      })
  },
})

export default productSlice.reducer
```

Phase 6: Key React Components

Step 11: Create Main Components

components/ProductList.js:

```
jsx
```

```
import React, { useEffect } from 'react'
import { useDispatch, useSelector } from 'react-redux'
import { fetchProducts } from '../store/slices/productSlice'
import ProductCard from './ProductCard'
import { Grid, Container, Typography } from '@mui/material'

const ProductList = () => {
  const dispatch = useDispatch()
  const { items: products, loading, error } = useSelector(state => state.products)

  useEffect(() => {
    dispatch(fetchProducts())
  }, [dispatch])

  if (loading) return <div>Loading...</div>
  if (error) return <div>Error: {error}</div>

  return (
    <Container>
      <Typography variant="h4" component="h1" gutterBottom>
        Products
      </Typography>
      <Grid container spacing={3}>
        {products.map(product => (
          <Grid item xs={12} sm={6} md={4} key={product.id}>
            <ProductCard product={product} />
          </Grid>
        ))}
      </Grid>
    </Container>
  )
}

export default ProductList
```

components/ProductCard.js:

```
jsx
```

```
import React from 'react'
import { useDispatch } from 'react-redux'
import { addToCart } from '../store/slices/cartSlice'
import {
  Card,
  CardMedia,
  CardContent,
  CardActions,
  Typography,
  Button,
  Box
} from '@mui/material'
import { ShoppingCart } from '@mui/icons-material'

const ProductCard = ({ product }) => {
  const dispatch = useDispatch()

  const handleAddToCart = () => {
    dispatch(addToCart({
      id: product.id,
      name: product.name,
      price: product.price,
      image: product.primaryImage,
      quantity: 1
    }))
  }

  return (
    <Card sx={{ height: '100%', display: 'flex', flexDirection: 'column' }}>
      <CardMedia
        component="img"
        height="240"
        image={product.primaryImage || '/placeholder-image.jpg'}
        alt={product.name}
      />
      <CardContent sx={{ flexGrow: 1 }}>
        <Typography gutterBottom variant="h6" component="h2">
          {product.name}
        </Typography>
        <Typography variant="body2" color="text.secondary">
          {product.shortDescription}
        </Typography>
        <Box sx={{ mt: 2 }}>
          <Typography variant="h6" color="primary">
            ${product.price}
          </Typography>
          {product.compareAtPrice && (
            <Typography
              variant="body2"
              color="text.secondary"
              sx={{ textDecoration: 'line-through' }}
            >
              ${product.compareAtPrice}
            </Typography>
          )}
        </Box>
      </CardContent>
      <CardActions>
        <Button
          size="small"
          variant="contained"
          startIcon={<ShoppingCart />}
          onClick={handleAddToCart}
          fullWidth
        >
          Add to Cart
        </Button>
      </CardActions>
    </Card>
  )
}

export default ProductCard
```

components/ShoppingCart.js:

```
jsx
```

```
import React from 'react'
import { useSelector, useDispatch } from 'react-redux'
import { removeFromCart, updateQuantity } from '../store/slices/cartSlice'
import {
  Drawer,
  Box,
  Typography,
  List,
  ListItem,
  ListItemText,
  ListItemAvatar,
  Avatar,
  IconButton,
  TextField,
  Button,
  Divider
} from '@mui/material'
import { Delete, Add, Remove } from '@mui/icons-material'

const ShoppingCart = ({ open, onClose }) => {
  const dispatch = useDispatch()
  const { items, totalAmount } = useSelector(state => state.cart)

  const handleQuantityChange = (id, newQuantity) => {
    if (newQuantity > 0) {
      dispatch(updateQuantity({ id, quantity: newQuantity }))
    }
  }

  const handleRemoveItem = (id) => {
    dispatch(removeFromCart(id))
  }

  return (
    <Drawer anchor="right" open={open} onClose={onClose}>
      <Box sx={{ width: 350, p: 2 }}>
        <Typography variant="h6" gutterBottom>
          Shopping Cart ({items.length} items)
        </Typography>
        <Divider />

        <List>
          {items.map((item) => (
            <ListItem key={item.id} sx={{ px: 0 }}>
              <ListItemAvatar>
                <Avatar src={item.image} alt={item.name} />
          </ListItemAvatar>
          <ListItemText
            primary={item.name}
            secondary={`${item.price}`}
          />
          <Box sx={{ display: 'flex', alignItems: 'center', gap: 1 }}>
            <IconButton
              size="small"
              onClick={() => handleQuantityChange(item.id, item.quantity - 1)}
            >
              <Remove />
            </IconButton>
            <TextField
              size="small"
              value={item.quantity}
              onChange={(e) => handleQuantityChange(item.id, parseInt(e.target.value))}
              sx={{ width: 60 }}
              inputProps={{ min: 1, type: 'number' }}
            />
            <IconButton
              size="small"
              onClick={() => handleQuantityChange(item.id, item.quantity + 1)}
            >
              <Add />
            </IconButton>
            <IconButton
              size="small"
              color="error"
              onClick={() => handleRemoveItem(item.id)}
            >
              <Delete />
            </IconButton>
          </Box>
        </ListItem>
      ))}
    </List>

    {items.length === 0 && (
      <Typography variant="body2" color="text.secondary" sx={{ textAlign: 'center', py: 4 }}>
        Your cart is empty
      </Typography>
    )}

    {items.length > 0 && (
      <>
        <Divider />
        <Box sx={{ mt: 2 }}>
```

```
            <Typography variant="h6">
             Total: ${totalAmount.toFixed(2)}
            </Typography>
            <Button
             variant="contained"
             fullWidth
             sx={{ mt: 2 }}
             size="large"
            >
             Checkout
            </Button>
           </Box>
          </>
        )}
      </Box>
    </Drawer>
  )
}

export default ShoppingCart
```

Phase 7: Authentication & User Management

Step 12: Authentication Service (Backend)

Services/IAuthService.cs in `Shopzy.Business`:

```csharp
public interface IAuthService
{
    Task<AuthResult> LoginAsync(LoginDto loginDto);
    Task<AuthResult> RegisterAsync(RegisterDto registerDto);
    Task<AuthResult> RefreshTokenAsync(string refreshToken);
    Task<bool> LogoutAsync(string userId);
}

public class AuthResult
{
    public bool Success { get; set; }
    public string Token { get; set; }
    public string RefreshToken { get; set; }
    public UserDto User { get; set; }
    public string Message { get; set; }
}
```

AuthService.cs:

```
csharp
```

```
public class AuthService : IAuthService
{
    private readonly IRepository<User> _userRepository;
    private readonly IPasswordHasher<User> _passwordHasher;
    private readonly IConfiguration _configuration;
    private readonly IMapper _mapper;

    public AuthService(
        IRepository<User> userRepository,
        IPasswordHasher<User> passwordHasher,
        IConfiguration configuration,
        IMapper mapper)
    {
        _userRepository = userRepository;
        _passwordHasher = passwordHasher;
        _configuration = configuration;
        _mapper = mapper;
    }

    public async Task<AuthResult> LoginAsync(LoginDto loginDto)
    {
        var users = await _userRepository.FindAsync(u => u.Email == loginDto.Email);
        var user = users.FirstOrDefault();

        if (user == null || !VerifyPassword(user, loginDto.Password))
        {
            return new AuthResult { Success = false, Message = "Invalid credentials" };
        }

        var token = GenerateJwtToken(user);
        var refreshToken = GenerateRefreshToken();

        return new AuthResult
        {
            Success = true,
            Token = token,
            RefreshToken = refreshToken,
            User = _mapper.Map<UserDto>(user)
        };
    }

    public async Task<AuthResult> RegisterAsync(RegisterDto registerDto)
    {
        var existingUsers = await _userRepository.FindAsync(u => u.Email == registerDto.Email);
        if (existingUsers.Any())
        {
```

```
        return new AuthResult { Success = false, Message = "Email already exists" };
    }

    var user = new User
    {
        Email = registerDto.Email,
        FirstName = registerDto.FirstName,
        LastName = registerDto.LastName,
        CreatedAt = DateTime.UtcNow,
        UpdatedAt = DateTime.UtcNow
    };

    user.PasswordHash = _passwordHasher.HashPassword(user, registerDto.Password);

    await _userRepository.AddAsync(user);
    await _userRepository.SaveChangesAsync();

    var token = GenerateJwtToken(user);
    return new AuthResult
    {
        Success = true,
        Token = token,
        User = _mapper.Map<UserDto>(user)
    };
}

private string GenerateJwtToken(User user)
{
    var tokenHandler = new JwtSecurityTokenHandler();
    var key = Encoding.UTF8.GetBytes(_configuration["Jwt:Key"]);
    var tokenDescriptor = new SecurityTokenDescriptor
    {
        Subject = new ClaimsIdentity(new[]
        {
            new Claim(ClaimTypes.NameIdentifier, user.Id.ToString()),
            new Claim(ClaimTypes.Email, user.Email),
            new Claim(ClaimTypes.Name, $"{user.FirstName} {user.LastName}")
        }),
        Expires = DateTime.UtcNow.AddHours(24),
        SigningCredentials = new SigningCredentials(new SymmetricSecurityKey(key), SecurityAlgorithms.HmacSha
        Issuer = _configuration["Jwt:Issuer"],
        Audience = _configuration["Jwt:Audience"]
    };

    var token = tokenHandler.CreateToken(tokenDescriptor);
    return tokenHandler.WriteToken(token);
}
```

```
    private bool VerifyPassword(User user, string password)
    {
        var result = _passwordHasher.VerifyHashedPassword(user, user.PasswordHash, password);
        return result == PasswordVerificationResult.Success;
    }

    private string GenerateRefreshToken()
    {
        var randomNumber = new byte[32];
        using var rng = RandomNumberGenerator.Create();
        rng.GetBytes(randomNumber);
        return Convert.ToBase64String(randomNumber);
    }
}
```

Step 13: Authentication Controller

AuthController.cs:

```
csharp
```

```
[ApiController]
[Route("api/[controller]")]
public class AuthController : ControllerBase
{
    private readonly IAuthService _authService;

    public AuthController(IAuthService authService)
    {
        _authService = authService;
    }

    [HttpPost("login")]
    public async Task<ActionResult<AuthResult>> Login(LoginDto loginDto)
    {
        var result = await _authService.LoginAsync(loginDto);
        if (!result.Success)
            return BadRequest(result);

        return Ok(result);
    }

    [HttpPost("register")]
    public async Task<ActionResult<AuthResult>> Register(RegisterDto registerDto)
    {
        var result = await _authService.RegisterAsync(registerDto);
        if (!result.Success)
            return BadRequest(result);

        return Ok(result);
    }

    [HttpPost("logout")]
    [Authorize]
    public async Task<ActionResult> Logout()
    {
        var userId = User.FindFirst(ClaimTypes.NameIdentifier)?.Value;
        await _authService.LogoutAsync(userId);
        return Ok(new { message = "Logged out successfully" });
    }
}
```

Step 14: Frontend Authentication

store/slices/authSlice.js:

```
javascript
```

```
import { createSlice, createAsyncThunk } from '@reduxjs/toolkit'
import api from '../../services/api'

export const login = createAsyncThunk(
  'auth/login',
  async ({ email, password }, { rejectWithValue }) => {
    try {
      const response = await api.post('/auth/login', { email, password })
      localStorage.setItem('token', response.data.token)
      return response.data
    } catch (error) {
      return rejectWithValue(error.response.data.message)
    }
  }
)

export const register = createAsyncThunk(
  'auth/register',
  async (userData, { rejectWithValue }) => {
    try {
      const response = await api.post('/auth/register', userData)
      localStorage.setItem('token', response.data.token)
      return response.data
    } catch (error) {
      return rejectWithValue(error.response.data.message)
    }
  }
)

const authSlice = createSlice({
  name: 'auth',
  initialState: {
    user: null,
    token: localStorage.getItem('token'),
    isLoading: false,
    error: null,
    isAuthenticated: false,
  },
  reducers: {
    logout: (state) => {
      localStorage.removeItem('token')
      state.user = null
      state.token = null
      state.isAuthenticated = false
    },
    clearError: (state) => {
      state.error = null
    },
  },
  extraReducers: (builder) => {
    builder
      .addCase(login.pending, (state) => {
        state.isLoading = true
        state.error = null
      })
      .addCase(login.fulfilled, (state, action) => {
        state.isLoading = false
        state.user = action.payload.user
        state.token = action.payload.token
        state.isAuthenticated = true
      })
      .addCase(login.rejected, (state, action) => {
        state.isLoading = false
        state.error = action.payload
      })
      .addCase(register.pending, (state) => {
        state.isLoading = true
        state.error = null
      })
      .addCase(register.fulfilled, (state, action) => {
        state.isLoading = false
        state.user = action.payload.user
        state.token = action.payload.token
        state.isAuthenticated = true
      })
      .addCase(register.rejected, (state, action) => {
        state.isLoading = false
        state.error = action.payload
      })
  },
})

export const { logout, clearError } = authSlice.actions
export default authSlice.reducer
```

components/LoginForm.js:

```jsx
```

```
import React from 'react'
import { useForm } from 'react-hook-form'
import { yupResolver } from '@hookform/resolvers/yup'
import * as yup from 'yup'
import { useDispatch, useSelector } from 'react-redux'
import { login } from '../store/slices/authSlice'
import {
  TextField,
  Button,
  Box,
  Typography,
  Alert,
  Paper,
  Container
} from '@mui/material'

const schema = yup.object({
  email: yup.string().email('Invalid email').required('Email is required'),
  password: yup.string().required('Password is required'),
})

const LoginForm = () => {
  const dispatch = useDispatch()
  const { isLoading, error } = useSelector(state => state.auth)

  const {
    register,
    handleSubmit,
    formState: { errors }
  } = useForm({
    resolver: yupResolver(schema)
  })

  const onSubmit = (data) => {
    dispatch(login(data))
  }

  return (
    <Container maxWidth="sm">
      <Paper elevation={3} sx={{ p: 4, mt: 4 }}>
        <Typography variant="h4" component="h1" gutterBottom textAlign="center">
          Login to Shopzy
        </Typography>

        {error && <Alert severity="error" sx={{ mb: 2 }}>{error}</Alert>}
        <Box component="form" onSubmit={handleSubmit(onSubmit)}>
          <TextField
            fullWidth
            margin="normal"
            label="Email"
            type="email"
            {...register('email')}
            error={!!errors.email}
            helperText={errors.email?.message}
          />

          <TextField
            fullWidth
            margin="normal"
            label="Password"
            type="password"
            {...register('password')}
            error={!!errors.password}
            helperText={errors.password?.message}
          />

          <Button
            type="submit"
            fullWidth
            variant="contained"
            sx={{ mt: 3, mb: 2 }}
            disabled={isLoading}
          >
            {isLoading ? 'Logging in...' : 'Login'}
          </Button>
        </Box>
      </Paper>
    </Container>
  )
}

export default LoginForm
```

Phase 8: Deployment Preparation

Step 15: Environment Configuration

appsettings.json (Backend):

```
json
```

```
{
  "ConnectionStrings": {
    "DefaultConnection": "Server=(localdb)\\mssqllocaldb;Database=ShopzyDB;Trusted_Connection=true;MultipleA
  },
  "Jwt": {
    "Key": "your-secret-key-here-make-it-long-and-secure",
    "Issuer": "Shopzy",
    "Audience": "Shopzy-Users",
    "ExpiresInHours": 24
  },
  "Logging": {
    "LogLevel": {
      "Default": "Information",
      "Microsoft.AspNetCore": "Warning"
    }
  },
  "AllowedHosts": "*"
}
```

package.json scripts (Frontend):

```json
{
  "scripts": {
    "start": "react-scripts start",
    "build": "react-scripts build",
    "test": "react-scripts test",
    "eject": "react-scripts eject"
  },
  "proxy": "https://localhost:5001"
}
```

Step 16: Docker Configuration (Optional)

Dockerfile (Backend):

```dockerfile
```

```
FROM mcr.microsoft.com/dotnet/aspnet:8.0 AS base
WORKDIR /app
EXPOSE 80
EXPOSE 443

FROM mcr.microsoft.com/dotnet/sdk:8.0 AS build
WORKDIR /src
COPY ["Shopzy.API/Shopzy.API.csproj", "Shopzy.API/"]
RUN dotnet restore "Shopzy.API/Shopzy.API.csproj"
COPY . .
WORKDIR "/src/Shopzy.API"
RUN dotnet build "Shopzy.API.csproj" -c Release -o /app/build

FROM build AS publish
RUN dotnet publish "Shopzy.API.csproj" -c Release -o /app/publish

FROM base AS final
WORKDIR /app
COPY --from=publish /app/publish .
ENTRYPOINT ["dotnet", "Shopzy.API.dll"]
```

Phase 9: Testing Strategy

Step 17: Unit Tests Setup

Install testing packages:

```bash
# Backend
dotnet add Shopzy.Tests package Microsoft.EntityFrameworkCore.InMemory
dotnet add Shopzy.Tests package xunit
dotnet add Shopzy.Tests package Moq

# Frontend
npm install --save-dev @testing-library/react @testing-library/jest-dom
```

Sample test (ProductServiceTests.cs):

```csharp
```

```
public class ProductServiceTests
{
    [Fact]
    public async Task GetProductByIdAsync_ReturnsProduct_WhenProductExists()
    {
        // Arrange
        var mockRepo = new Mock<IRepository<Product>>();
        var product = new Product { Id = 1, Name = "Test Product" };
        mockRepo.Setup(r => r.GetByIdAsync(1)).ReturnsAsync(product);

        var service = new ProductService(mockRepo.Object, null);

        // Act
        var result = await service.GetProductByIdAsync(1);

        // Assert
        Assert.NotNull(result);
        Assert.Equal("Test Product", result.Name);
    }
}
```

Phase 10: Next Steps & Advanced Features

Recommended Implementation Order:

1. **Phase 1-4**: Core backend setup (Database, Models, Services, Basic API)
2. **Phase 5-6**: Basic React frontend (Product listing, Cart functionality)
3. **Phase 7**: Authentication system
4. **Phase 8**: Deployment preparation
5. **Phase 9**: Testing implementation

Advanced Features to Add Later:

- **Payment Integration** (Stripe, PayPal)
- **Email Services** (Order confirmations, newsletters)
- **File Upload** (Product images)
- **Search & Filtering** (Elasticsearch integration)
- **Admin Dashboard** (Product management, Order management)
- **Inventory Management**
- **Multi-vendor Support**
- **Reviews & Ratings System**

- **Wishlist Functionality**
- **Real-time Notifications** (SignalR)
- **Mobile App** (React Native)

Security Considerations:

- Input validation and sanitization
- CORS configuration
- Rate limiting
- SQL injection prevention
- XSS protection
- CSRF tokens
- Secure password policies
- HTTPS enforcement

This guide provides a solid foundation for your Shopzy e-commerce application. Start with the basic functionality and gradually add more features as needed.