BookaBarber - Barber Shop Booking System

A comprehensive web application for booking barber services, connecting customers with skilled barbers in their area. BookaBarber streamlines appointment scheduling while providing powerful management tools for barbers and shop owners.

▯ Table of Contents

▯ Features

Customer Features

- **User Profile Management**
  - Personal information storage with secure data handling
  - Profile photo upload with image optimization
  - Preference settings for notifications and communications
  - Service history tracking and analytics
  - Favorite barbers list for quick booking
- **Location-Based Services**
  - Geolocation API integration for current location detection
  - Radius-based search with adjustable parameters
  - Map view with interactive barber shop pins
  - Route generation to selected barber shop
  - Location-based recommendations
- **Service Selection & Booking**
  - Categorized service menu with detailed descriptions
  - Visual gallery of hairstyle options
  - Custom service requests with pricing estimates
  - Multiple service selection for combination bookings
  - Quick rebooking from service history
- **Appointment Management**
  - Interactive calendar with availability highlighting
  - Time slot selection with duration indicators
  - Real-time booking confirmation
  - Appointment modification with business rules enforcement
  - Cancellation system with configurable policies
- **User Experience**
  - Dark/light theme with system preference detection
  - Responsive design for all device types
  - Accessibility features (WCAG compliance)
  - Multi-language support with auto-detection
  - Advanced search with filters and sorting options

Barber/Shop Features

- **Business Profile Management**
  - Customizable shop profile with rich media gallery
  - Staff management with individual barber profiles
  - Service catalog with pricing tier options
  - Operating hours with exception date handling
  - Business analytics dashboard
- **Scheduling & Calendar Management**

- **Scheduling & Calendar Management**
    - Multi-view calendar (day, week, month)
    - Resource allocation for multiple barbers
    - Buffer time configuration between appointments
    - Vacation and time-off planning
    - Recurring booking handling
- **Customer Relationship Management**
    - Customer database with service history
    - Notes and preferences tracking
    - Communication tools for direct messaging
    - Customer segmentation for targeted promotions
    - Loyalty program management
- **Financial Tools**
    - Revenue tracking and reporting
    - Service-based performance analytics
    - Commission calculation for barbers
    - Tax report generation
    - Payment reconciliation tools

Admin Features

- **System Management**
    - User account administration with role-based access
    - Global settings configuration
    - System health monitoring
    - Database maintenance tools
    - Activity logging and audit trails
- **Content Management**
    - News and announcements publication
    - FAQ and help documentation editor
    - Email template customization
    - Marketing campaign management
    - Terms of service and policy administration
- **Analytics & Reporting**
    - Cross-shop performance comparisons
    - Trend analysis and forecasting
    - Custom report generation
    - Data export in multiple formats
    - Real-time dashboard with KPIs

□ Technology Stack

Frontend

- **Core Technologies**
    - HTML5 with semantic markup
    - CSS3 with Flexbox and Grid layouts
    - JavaScript (ES6+) with async/await patterns
    - Bootstrap 5 framework for responsive design
- **UI/UX Enhancements**
    - FontAwesome 6 for vector icons
    - Custom CSS animations and transitions
    - Interactive charts with Chart.js
    - Image lazy loading and optimization
    - Progressive Web App capabilities

Backend

- **Core Framework**
    - Python 3.8+ with type hints
    - Flask web framework with Blueprints architecture
    - SQLAlchemy ORM for database interactions
    - Jinja2 templating engine
    - RESTful API design principles
- **Authentication & Security**
    - JWT (JSON Web Tokens) for stateless authentication
    - Role-based access control (RBAC)
    - Password hashing with bcrypt
    - CSRF protection
    - Rate limiting for API endpoints

Database

- **Primary Database**
    - SQLite for development
    - PostgreSQL recommended for production
    - Database migration management with Alembic
    - Connection pooling
    - Query optimization

Services

- **Email System**
    - SMTP integration with failover
    - Template-based email generation
    - Queue-based sending for reliability
    - Delivery status tracking
    - Bounce handling
- **Geolocation Services**
    - OpenStreetMap integration
    - Geocoding and reverse geocoding
    - Distance calculation algorithms
    - Location data caching
    - Boundary detection for service areas
- **Analytics & Monitoring**
    - Custom event tracking
    - User behavior analysis
    - Performance metric collection
    - Error logging and alerting
    - A/B testing framework

□ System Architecture

MVC Pattern Implementation

- **Model Layer**: Data structures and business logic
- **View Layer**: Template-based UI rendering
- **Controller Layer**: Request handling and response generation

Service Oriented Architecture

- Decoupled services with clear interfaces
- Microservices approach for core functions:
    - Booking Service
    - Notification Service
    - User Management Service
    - Analytics Service

Request Flow

```
 Client Request → Routing → Authentication → Authorization → Controller → Service Layer →
Data Access Layer → Database → Response Generation → Client
```

Caching Strategy

- Multi-level caching:
    - Browser-level caching
    - Application-level memory cache
    - Redis-based distributed cache
- Invalidation policies for data consistency

Asynchronous Processing

- Task queue implementation with Celery
- Background job processing for:
    - Email sending
    - Notification delivery
    - Report generation
    - Data synchronization

□ Installation

System Requirements

- **Operating System**: Linux, macOS, or Windows
- **CPU**: Dual-core 2.0 GHz or higher
- **RAM**: Minimum 4GB (8GB recommended)
- **Storage**: 1GB available space for application
- **Network**: Broadband internet connection

Prerequisites

- Python 3.8 or higher
- pip (Python package manager)
- Git
- Node.js and npm (for frontend asset management)
- Redis (optional, for advanced caching)

Clone Repository

```
 # Clone the repository
git clone https://github.com/Lusan-sapkota/Barber-booking-system.git

# Navigate to project directory
cd Barber-shop-booking-system
```

Setup Environment

```
 # Create virtual environment
python -m venv venv

# Activate virtual environment
# On Windows
venv\Scripts\activate
# On macOS/Linux
source venv/bin/activate

# Install Python dependencies
pip install -r requirements.txt

# Install frontend dependencies (optional)
npm install
```

Initialize Database

```
 # Initialize SQLite database with schema
python -c "from models import db; db.create_all()"

# Run database migrations
python manage.py db upgrade

# Seed initial data (optional)
python manage.py seed
```

Verify Installation

```
 # Run tests to verify setup
python -m pytest

# Start development server
python app.py
```

Database Configuration

The system supports both SQLite (for development) and PostgreSQL (recommended for production) databases.

SQLite (Default)

SQLite is configured by default and requires no additional setup:

```
# Initialize SQLite database with schema
python -c "from models import db; db.create_all()"

# Run database migrations
python manage.py db upgrade

# Seed initial data (optional)
python manage.py seed
```

PostgreSQL

For production environments, we recommend using PostgreSQL:

1. Install PostgreSQL and Required Python Packages

```
# Install PostgreSQL (Ubuntu/Debian)
sudo apt-get update
sudo apt-get install postgresql postgresql-contrib

# Install Python packages
pip install psycopg2-binary python-dotenv
```

2. Create a PostgreSQL Database

```
# Login to PostgreSQL
sudo -u postgres psql

# Create database and user
CREATE DATABASE barbershop;
CREATE USER barbershop_user WITH ENCRYPTED PASSWORD 'your_secure_password';
GRANT ALL PRIVILEGES ON DATABASE barbershop TO barbershop_user;

# Exit PostgreSQL
\q
```

3. Create Environment Variables

Create a .env file in your project root:

```
# PostgreSQL Configuration
POSTGRES_HOST=localhost
POSTGRES_DB=barbershop
POSTGRES_USER=barbershop_user
POSTGRES_PASSWORD=your_secure_password
POSTGRES_PORT=5432
```

4. Initialize Database

Create a script called `init_postgres_db.py`:

```
from models_postgres import init_db

print("Initializing PostgreSQL database...")
init_db()
print("Database initialization complete!")
```

Run it to create all tables:

```
python init_postgres_db.py
```

5. Update Your Application to Use PostgreSQL

Modify your app.py to use the PostgreSQL models:

```
# Change this line:
from models import (init_db, Booking, Barber, Customer, Service, User, Shop,
                   Notification, AdminAction, SystemSettings, Review)

# To this:
from models_postgres import (init_db, Booking, Barber, Customer, Service, User, Shop,
                            Notification, AdminAction, SystemSettings, Review)
```

6. Migrate Existing Data (Optional)

If you need to migrate data from SQLite to PostgreSQL, create a migration script (see original README for full script).

7. PostgreSQL Performance Optimization

For optimal PostgreSQL performance:

```
-- Create indexes for frequently queried fields
CREATE INDEX idx_bookings_date ON bookings(date);
CREATE INDEX idx_barbers_shop_id ON barbers(shop_id);

-- Add full text search capability
CREATE EXTENSION pg_trgm;
CREATE INDEX barbers_name_search_idx ON barbers USING gin(name gin_trgm_ops);
CREATE INDEX services_name_search_idx ON services USING gin(name gin_trgm_ops);
```

8. Connection Pooling with PgBouncer (Production)

For high-traffic production environments:

```
# Install PgBouncer
sudo apt-get install pgbouncer

# Configure PgBouncer
# Edit /etc/pgbouncer/pgbouncer.ini

# Example configuration:
[databases]
barbershop = host=localhost port=5432 dbname=barbershop

[pgbouncer]
listen_port = 6432
listen_addr = 0.0.0.0
auth_type = md5
auth_file = /etc/pgbouncer/userlist.txt
pool_mode = transaction
max_client_conn = 1000
default_pool_size = 20
```

⚙ Configuration

Environment Variables

Create a .env file in the root directory with the following variables:

```
# Application Settings
APP_NAME=BookaBarber
ENVIRONMENT=development  # development, testing, production
DEBUG=True
LOG_LEVEL=DEBUG

# Database Configuration
DATABASE_URL=sqlite:///barbershop.db
POOL_SIZE=10
MAX_OVERFLOW=20
POOL_TIMEOUT=30
POOL_RECYCLE=1800

# Security Settings
SECRET_KEY=your_secret_key
JWT_SECRET_KEY=your_jwt_secret
JWT_ACCESS_TOKEN_EXPIRES=3600  # seconds
PASSWORD_SALT=your_password_salt
ALLOWED_HOSTS=localhost,127.0.0.1

# Email Configuration
MAIL_SERVER=smtp.example.com
MAIL_PORT=587
MAIL_USERNAME=your_email@example.com
MAIL_PASSWORD=your_password
MAIL_USE_TLS=True
MAIL_DEFAULT_SENDER=noreply@bookabarber.com
MAIL_MAX_EMAILS=100

# Redis Configuration (Optional)
REDIS_URL=redis://localhost:6379/0
CACHE_TYPE=redis
CACHE_REDIS_URL=redis://localhost:6379/1

# Location Services
MAPS_API_KEY=your_maps_api_key
DEFAULT_SEARCH_RADIUS=10  # km
GEOCODING_CACHE_TIMEOUT=86400  # seconds

# Feature Flags
ENABLE_SOCIAL_LOGIN=True
ENABLE_DYNAMIC_PRICING=True
ENABLE_NOTIFICATIONS=True
ENABLE_ANALYTICS=True
```

Configuration Files

- **config.py**: Core configuration file with environment-specific settings
- **logging.ini**: Logging configuration
- **gunicorn.conf.py**: Production server settings

Email Configuration

For development, you can use sandbox environments:

Mailtrap

```
MAIL_SERVER=smtp.mailtrap.io
MAIL_PORT=2525
MAIL_USERNAME=your_mailtrap_username
MAIL_PASSWORD=your_mailtrap_password
MAIL_USE_TLS=True
```

Local Testing with Mailhog

```
MAIL_SERVER=localhost
MAIL_PORT=1025
MAIL_USERNAME=
MAIL_PASSWORD=
```

SendGrid (Production)

```
MAIL_SERVER=smtp.sendgrid.net
MAIL_PORT=587
MAIL_USERNAME=apikey
MAIL_PASSWORD=your_sendgrid_api_key
MAIL_USE_TLS=True
```

Third-Party Integrations

Payment Processors

- Stripe configuration:

  ```
  STRIPE_PUBLIC_KEY=pk_test_...
  STRIPE_SECRET_KEY=sk_test_...
  STRIPE_WEBHOOK_SECRET=whsec_...
  ```

- PayPal configuration:

  ```
  PAYPAL_CLIENT_ID=client_id...
  PAYPAL_CLIENT_SECRET=client_secret...
  PAYPAL_MODE=sandbox  # or 'live'
  ```

Social Authentication

- Google OAuth:

  ```
  GOOGLE_CLIENT_ID=your_client_id
  GOOGLE_CLIENT_SECRET=your_client_secret
  GOOGLE_DISCOVERY_URL=https://accounts.google.com/.well-known/openid-configuration
  ```

- Facebook OAuth:

  ```
  FACEBOOK_CLIENT_ID=your_app_id
  FACEBOOK_CLIENT_SECRET=your_app_secret
  ```

� Algorithms & Technical Implementation

Appointment Scheduling Algorithm

- **Implementation**: algorithms.py - `SchedulingAlgorithm` class
- **Core Algorithm**: Priority queue-based resource allocation
- **Constraints Handling**:
  - Barber availability windows
  - Service duration requirements
  - Travel time between appointments
  - Barber specialization matching
  - Client preferences

Barber Recommendation Engine

- **Implementation**: algorithms.py - `RecommendationEngine` class
- **Core Algorithm**: Multi-factor weighted scoring with geographic filtering
- **Features Used**:
  - Geographic proximity using Haversine formula
  - Service-specific expertise and specialization
  - Rating trends and review sentiment analysis
  - Historical booking patterns and customer affinity

Smart Notification System

- **Implementation**: email_service.py - `NotificationManager` class
- **Core Algorithm**: Time-based event triggering with priority queueing
- **Notification Strategies**:
    - Progressive notification sequence (email → SMS → push)
    - Smart retry mechanism with exponential backoff
    - Delivery time optimization
    - Template personalization

Dynamic Pricing Model

- **Implementation**: algorithms.py - `DynamicPricingEngine` class
- **Core Algorithm**: Multi-variable regression with seasonal adjustment
- **Pricing Factors**:
    - Time-based adjustments (peak hours, weekends)
    - Barber experience and popularity metrics
    - Historical demand patterns by time slot
    - Seasonal adjustments (holidays, special events)

Graph-Based Service Recommendation

- **Implementation**: algorithms.py - `ServiceGraph` class
- **Core Algorithm**: Association rule mining with graph traversal
- **Application Areas**:
    - Bundle recommendations for service packages
    - Upselling opportunities identification
    - Personalized service discovery
    - Seasonal package optimization

□ Database Schema

Core Tables

- **Users**
    - Authentication credentials
    - Profile information
    - Account settings
    - Role assignments
- **Customers**
    - Personal information
    - Contact details
    - Preference settings
    - Service history
- **Barbers**
    - Professional information
    - Skills and specializations
    - Availability schedule
    - Performance metrics
- **Shops**
    - Business information
    - Location data
    - Operating hours
    - Staff associations
- **Services**
    - Service definitions
    - Pricing information
    - Duration estimates
    - Category classifications
- **Bookings**
    - Appointment details
    - Status tracking
    - Payment information
    - Related services

Relationship Diagram

```
User ─┬── Customer ──┬── Booking ──── Service
      └── Barber ────┘      │
                            │
                    Shop ───┘
```

Indexes and Optimizations

- Composite indexes for frequent query patterns
- Full-text search indexes for name and location searches
- Temporal partitioning for historical booking data
- Spatial indexing for location-based queries

□ Usage Guide

Running the Application

```
# Development mode
python app.py

# Production mode with gunicorn
gunicorn "app:create_app()" --bind 0.0.0.0:5000 --workers=4 --threads=2
```

Then open your browser and navigate to `http://127.0.0.1:5000`

Customer Flow

1. **Create Account / Sign In**
2. **Find a Barber**
3. **Book an Appointment**
4. **Manage Bookings**

Barber/Shop Flow

1. **Professional Registration**
2. **Manage Calendar**
3. **Process Appointments**
4. **Business Management**

Admin Flow

1. **System Management**
2. **Content Management**
3. **Analytics & Reporting**

□ API Documentation

API Overview

The BookaBarber API is a RESTful interface allowing programmatic access to the system's functionality. All endpoints return JSON responses.

Base URL

```
Development: http://localhost:5000/api
Production: https://api.bookabarber.com/v1
```

Authentication

All API requests require authentication using JSON Web Tokens (JWT):

```
Authorization: Bearer <your_jwt_token>
```

Key Endpoints

- **Authentication**: `/api/auth/register`, `/api/auth/login`
- **Barber Management**: `/api/barbers`, `/api/barbers/nearby`
- **Booking Management**: `/api/bookings`, `/api/bookings/{id}`
- **Service Management**: `/api/services`
- **Admin Functions**: `/api/admin/users`

□ Testing

Test Suite Structure

- **Unit Tests**: Test individual functions and classes
- **Integration Tests**: Test component interactions
- **API Tests**: Test API endpoints
- **End-to-End Tests**: Test complete user flows
- **Performance Tests**: Test system under load

Running Tests

```
 # Run all tests
pytest

# Run specific test category
pytest tests/unit/
pytest tests/api/

# Run with coverage report
pytest --cov=. tests/
```

Continuous Integration

Tests are automatically run on:

- Pull request creation
- Merge to main branch
- Daily scheduled runs

□ Security Considerations

Authentication Security

- Password hashing using bcrypt with salt
- JWT with short expiration and refresh token rotation
- Multi-factor authentication option
- Account lockout after failed attempts

Data Protection

- Encryption of sensitive data at rest
- TLS/SSL for all communications
- Regular security audits
- GDPR and CCPA compliance measures

API Security

- Rate limiting to prevent abuse
- Input validation and sanitization
- CSRF protection
- CORS policy configuration

Infrastructure Security

- Regular security patches
- Network segmentation
- Web Application Firewall (WAF)
- DDoS protection

□ Performance Optimization

Database Optimization

- Query optimization with proper indexing
- Connection pooling
- Statement caching
- Read replicas for scale

Caching Strategy

- Multi-level caching approach:
  - Browser caching for static assets
  - CDN for media content

    - Redis for application data
    - Memory cache for frequent lookups

Frontend Performance

- Asset bundling and minification
- Lazy loading of components
- Image optimization
- Critical CSS inlining

Backend Efficiency

- Asynchronous processing for long-running tasks
- Horizontal scaling for API services
- Optimized algorithms for core functions
- Response compression

▯ Deployment

Development Environment

- Local development with Docker
- Hot-reloading for rapid iteration
- Development database seeding
- Local email trapping

Staging Environment

- Cloud-based replica of production
- Integration with CI/CD pipeline
- Automated testing before promotion
- Data anonymization from production

Production Deployment

- Blue-green deployment strategy
- Automated rollback capabilities
- Health checks and monitoring
- Load balancing across multiple instances

Deployment Commands

```
# Build Docker image
docker build -t bookabarber:latest .

# Run Docker container
docker run -p 5000:5000 -e ENVIRONMENT=production bookabarber:latest

# Deploy to production
./deploy.sh production
```

▯ Troubleshooting

Common Issues

Application Won't Start

- Check database connection configuration
- Verify required environment variables are set
- Ensure Python version compatibility (3.8+)
- Check logs for specific error messages

Booking Creation Fails

- Verify selected time slot availability
- Check service and barber IDs validity
- Ensure user authentication
- Confirm required fields are provided

Email Notifications Not Sending

- Check SMTP server configuration
- Verify email templates exist
- Check email service logs for errors

PostgreSQL-Specific Issues

Connection Failures

- Verify PostgreSQL is running (`sudo systemctl status postgresql`)
- Check connection parameters in .env file
- Ensure database/user permissions are correct
- Confirm firewall allows connections to PostgreSQL port

Migration Errors

- Check column compatibility between SQLite and PostgreSQL
- Handle data type differences (BOOLEAN vs INTEGER)
- Ensure foreign key constraints are satisfied

Performance Issues

- Verify indexing on frequently queried columns
- Check query execution plans with `EXPLAIN ANALYZE`
- Monitor connection count and resource usage

Logging and Debugging

```
# Enable debug mode
export DEBUG=True

# Set verbose logging
export LOG_LEVEL=DEBUG

# Check application logs
tail -f logs/application.log

# Check error logs
tail -f logs/error.log
```

Contributing

We welcome contributions! Please follow these steps:

1. **Fork the Repository**

   ```
   git clone https://github.com/Lusan-sapkota/Barber-booking-system.git
   cd Barber-shop-booking-system
   ```

2. **Create a Feature Branch**

   ```
   git checkout -b feature/your-feature-name
   ```

3. **Set Up Development Environment**

   ```
   pip install -r requirements-dev.txt
   pre-commit install
   ```

4. **Make Your Changes**
   - Follow existing code style and conventions
   - Add tests for new functionality
   - Update documentation as needed

5. **Run Tests**

   ```
   pytest
   flake8
   ```

6. **Commit and Push**

```
 git add .
git commit -m 'Add some feature'
git push origin feature/your-feature-name
```

7. **Create a Pull Request**
    - Open a PR against the main repository
    - Provide clear description of changes
    - Reference related issues

Contribution Guidelines

- Follow existing code style and conventions
- Write clear, descriptive commit messages
- Include tests for new features and bug fixes
- Update documentation for API changes
- Keep PRs focused on single changes
- Ensure all tests pass before submitting

□ Roadmap

Upcoming Features (Q3 2023)

- Mobile application for iOS and Android
- Integrated payment processing
- AI-powered style recommendation
- Video consultation before booking
- Loyalty program with rewards

Medium Term (Q4 2023)

- Inventory management for shops
- Staff performance analytics
- Advanced reporting dashboard
- Customer retention tools
- Multi-language support

Long Term (2024)

- Marketplace for barber products
- Franchise management system
- Integrated POS system
- Machine learning for improved recommendations
- White-label solution for enterprise customers

□ License

This project is licensed under the Apache License 2.0 - see the LICENSE file for details.

The project also includes a custom license with additional terms - see LICENSE-CUSTOM.txt for specific requirements and permissions beyond the Apache License 2.0.

Created with ❤ by Lusan Sapkota. For issues, feature requests, or questions, please open an issue.