# BlogSite - Full-Stack Microservices Blogging Platform

[BlogSite - Full-Stack Microservices Blogging Platform 1](#_Toc220846508)

[1. Project Overview 1](#_Toc220846509)

[2. System Architecture 2](#_Toc220846510)

[3. Technology Stack 3](#_Toc220846511)

[4. Key Features 4](#_Toc220846512)

[5. Implementation Details 4](#_Toc220846513)

[6. Screenshots 5](#_Toc220846514)

[7. Setup and Execution 7](#_Toc220846515)

[8. Project Structure 9](#_Toc220846516)

[9. Security Features 10](#_Toc220846517)

[10. Project Zip Files 11](#_Toc220846518)

## 1. Project Overview

**Project Name:** BlogSite - Microservices Blogging Platform   
**Student Name:** Manvendra Pratap Singh  
**Employ ID:** 2252951

### Project Description

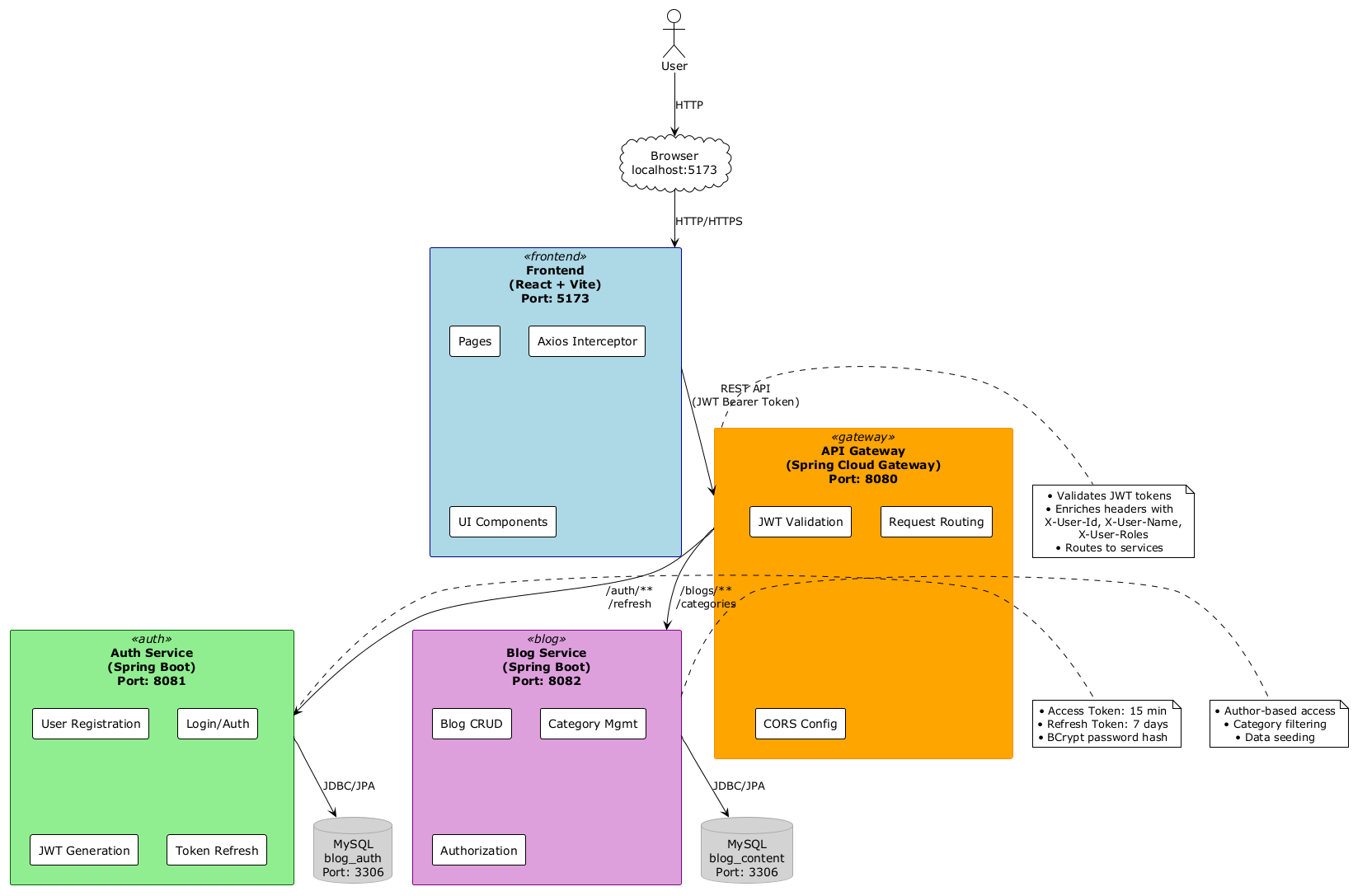
BlogSite is a full-stack web application built using microservices architecture. It allows users to create, read, update, and delete blog posts with authentication and authorization mechanisms. The project demonstrates industry-standard practices including JWT authentication with automatic token refresh, API Gateway pattern, and modern frontend development.

### Problem Statement

Develop a scalable blogging platform where: - Users can register and authenticate securely - Authenticated users can create and manage their blog posts - Blog posts are organized by categories - The system should handle authentication efficiently without constant re-login - The architecture should be scalable and maintainable

## 2. System Architecture

### Microservices Architecture Diagram



### Components Description

#### 2.1 Frontend (React + Vite)

* Modern single-page application (SPA)
* Communicates with backend through API Gateway
* Handles client-side routing and state management
* **Port:** 5173

#### 2.2 API Gateway (Spring Cloud Gateway)

* Single entry point for all client requests
* Routes requests to appropriate microservices
* JWT validation and authentication
* CORS configuration
* **Port:** 8080

#### 2.3 Auth Service

* User registration and login
* JWT token generation (Access + Refresh tokens)
* Token refresh mechanism
* User management
* **Port:** 8081
* **Database:** blog\_auth

#### 2.4 Blog Service

* Blog post CRUD operations
* Category management
* Author-based access control
* **Port:** 8082
* **Database:** blog\_content

#### 2.5 Database Layer

* **MySQL 8.0** - Two separate databases for service isolation
  + blog\_auth - User authentication data
  + blog\_content - Blog posts and categories

## 3. Technology Stack

### Backend Technologies

| Technology | Version | Purpose |
| --- | --- | --- |
| Java | 21 | Core Programming Language |
| Spring Boot | 3.2.0 | Application Framework |
| Spring Cloud Gateway | 4.1.0 | API Gateway (Reactive) |
| Spring Security | 6.2.0 | Authentication & Authorization |
| Spring Data JPA | 3.2.0 | Data Persistence Layer |
| Hibernate | 6.4.0 | ORM Framework |
| MySQL | 8.0 | Relational Database |
| JJWT | 0.11.5 | JWT Token Management |
| Maven | 3.8+ | Build & Dependency Management |

### Frontend Technologies

| Technology | Version | Purpose |
| --- | --- | --- |
| React | 19.2.0 | UI Library |
| Vite | 7.3.1 | Build Tool & Dev Server |
| Axios | 1.13.4 | HTTP Client with Interceptors |
| Tailwind CSS | 4.1.18 | Utility-first CSS Framework |
| Lucide React | 0.563.0 | Icon Library |
| shadcn/ui | Latest | Pre-built UI Components |

## 4. Key Features

### 4.1 Authentication & Authorization

✅ User registration with role assignment   
✅ Secure login with JWT tokens   
✅ Dual token system (Access token: 15 min, Refresh token: 7 days)   
✅ Automatic token refresh using Axios interceptors   
✅ Seamless user experience (no forced re-login for 7 days)   
✅ Password encryption using BCrypt

### 4.2 Blog Management

✅ Create blog posts with title, category, and content   
✅ View all blogs with category filtering   
✅ Update own blog posts   
✅ Delete own blog posts   
✅ Author-based access control   
✅ Pre-seeded categories and sample blogs

### 4.3 User Interface

✅ Responsive design (mobile, tablet, desktop)   
✅ Modern dark theme with Tailwind CSS   
✅ Toast notifications for user feedback   
✅ Loading states and error handling   
✅ Clean and intuitive navigation

### 4.4 Technical Excellence

✅ Microservices architecture   
✅ API Gateway pattern   
✅ Comprehensive exception handling   
✅ Database seeding for demo purposes   
✅ CORS configuration   
✅ RESTful API design

## 5. Implementation Details

### 5.1 JWT Authentication Flow

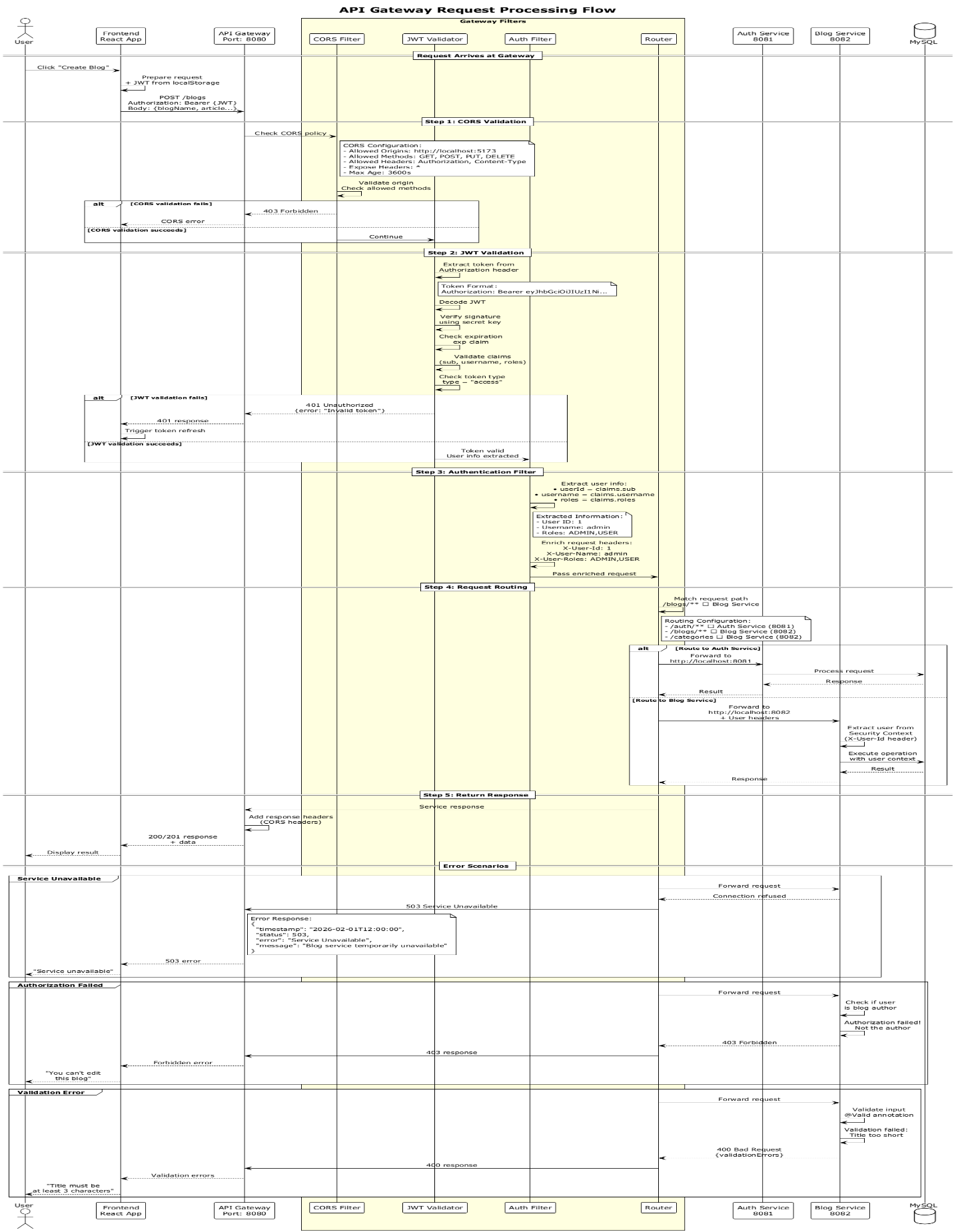
#### Token Generation Process

1. User submits credentials (username/password)
2. Auth service validates credentials
3. Upon success, generates two tokens:
   * **Access Token**: Short-lived (15 minutes), used for API requests
   * **Refresh Token**: Long-lived (7 days), used to obtain new access tokens
4. Both tokens stored in localStorage on client side

#### Automatic Token Refresh

* Axios interceptor monitors all API responses
* On receiving 401 Unauthorized:
  1. Attempts token refresh using refresh token
  2. Retries original failed request with new access token
  3. If refresh fails, redirects to login
* User experiences seamless authentication without interruption

### 5.2 API Gateway Request Flow



#### Request Processing Steps

1. **Client Request** → API Gateway receives request
2. **JWT Validation** → GatewayAuthenticationFilter validates JWT token
3. **Header Enrichment** → Gateway adds X-User-Id, X-User-Name, X-User-Roles headers
4. **Routing** → Routes to appropriate microservice (auth/blog)
5. **Service Processing** → Microservice uses Spring Security context
6. **Response** → Gateway returns response to client

### 5.3 Exception Handling Strategy

#### Auth Service Exception Handler

@RestControllerAdvice  
- IllegalArgumentException → 400 Bad Request  
- BadCredentialsException → 401 Unauthorized  
- MethodArgumentNotValidException → 400 with field errors  
- Generic Exception → 500 Internal Server Error

#### Blog Service Exception Handler

@RestControllerAdvice  
- IllegalArgumentException → 400 Bad Request  
- MethodArgumentNotValidException → 400 with validation errors  
- Generic Exception → 500 Internal Server Error

#### API Gateway Exception Handler

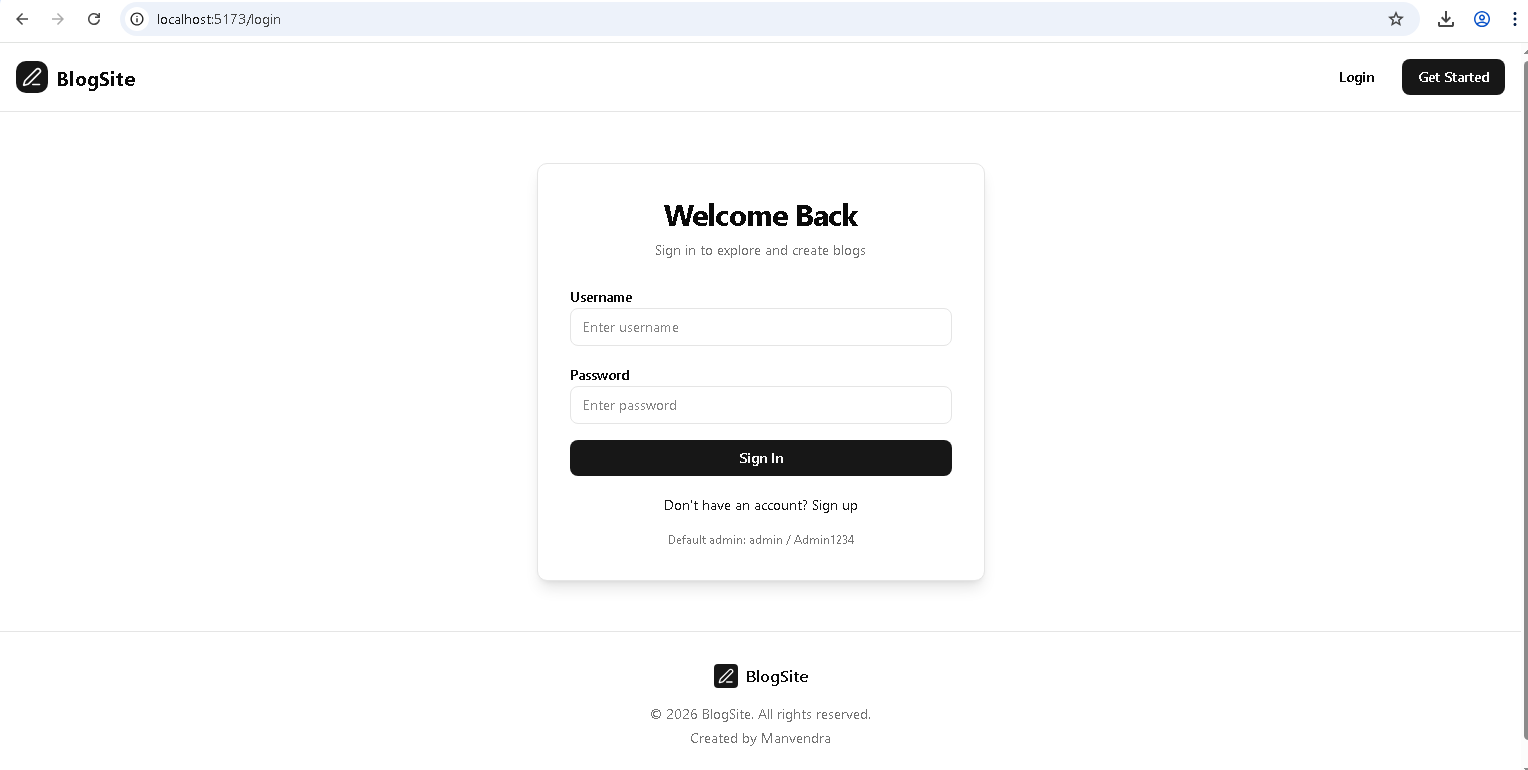
WebExceptionHandler (Reactive)  
- JWT errors → 401 Unauthorized  
- Connection refused → 503 Service Unavailable  
- Generic errors → 500 Internal Server Error

## 

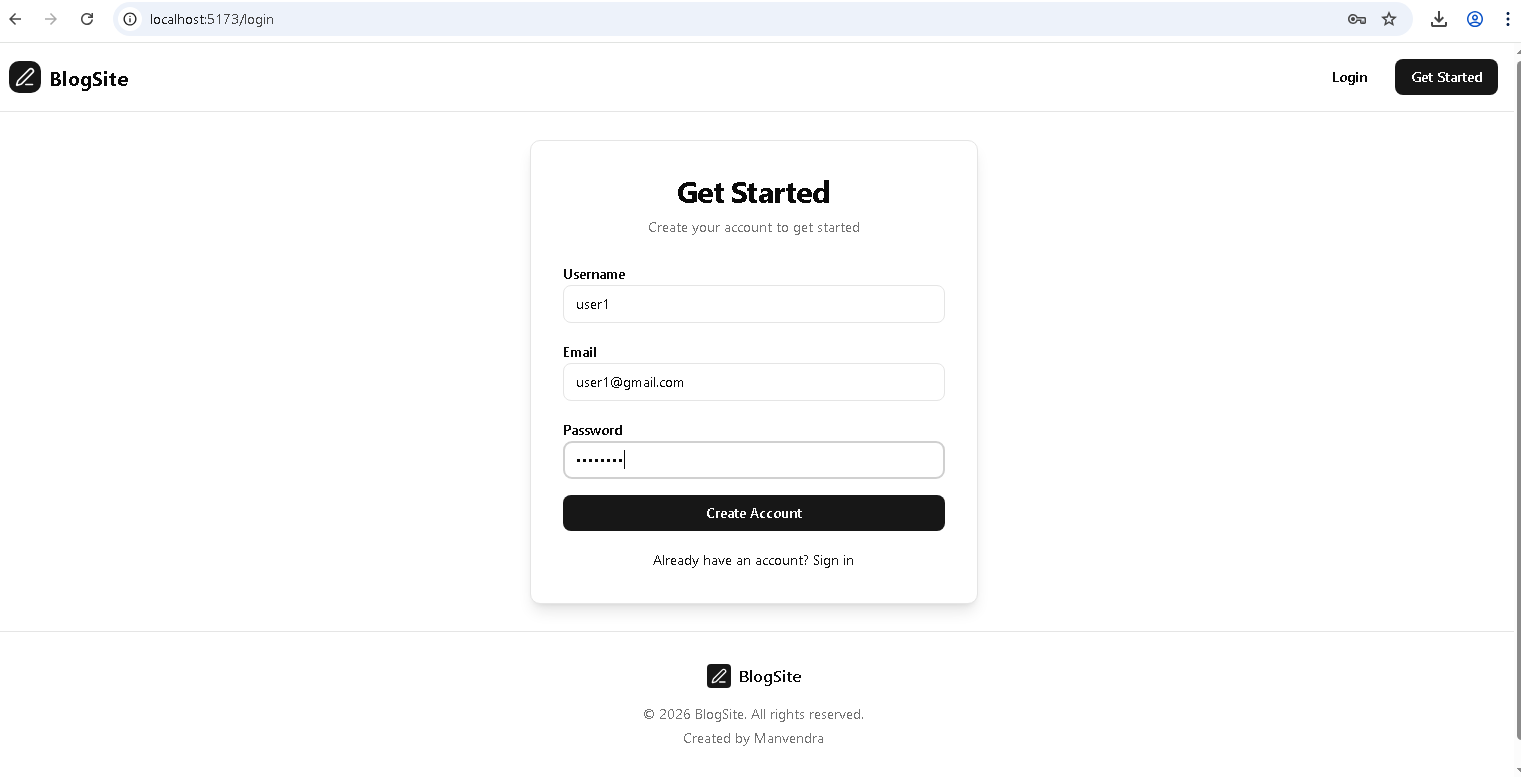
## 6. Screenshots

### 6.1 User Authentication

**Screenshot 1: Login Page**



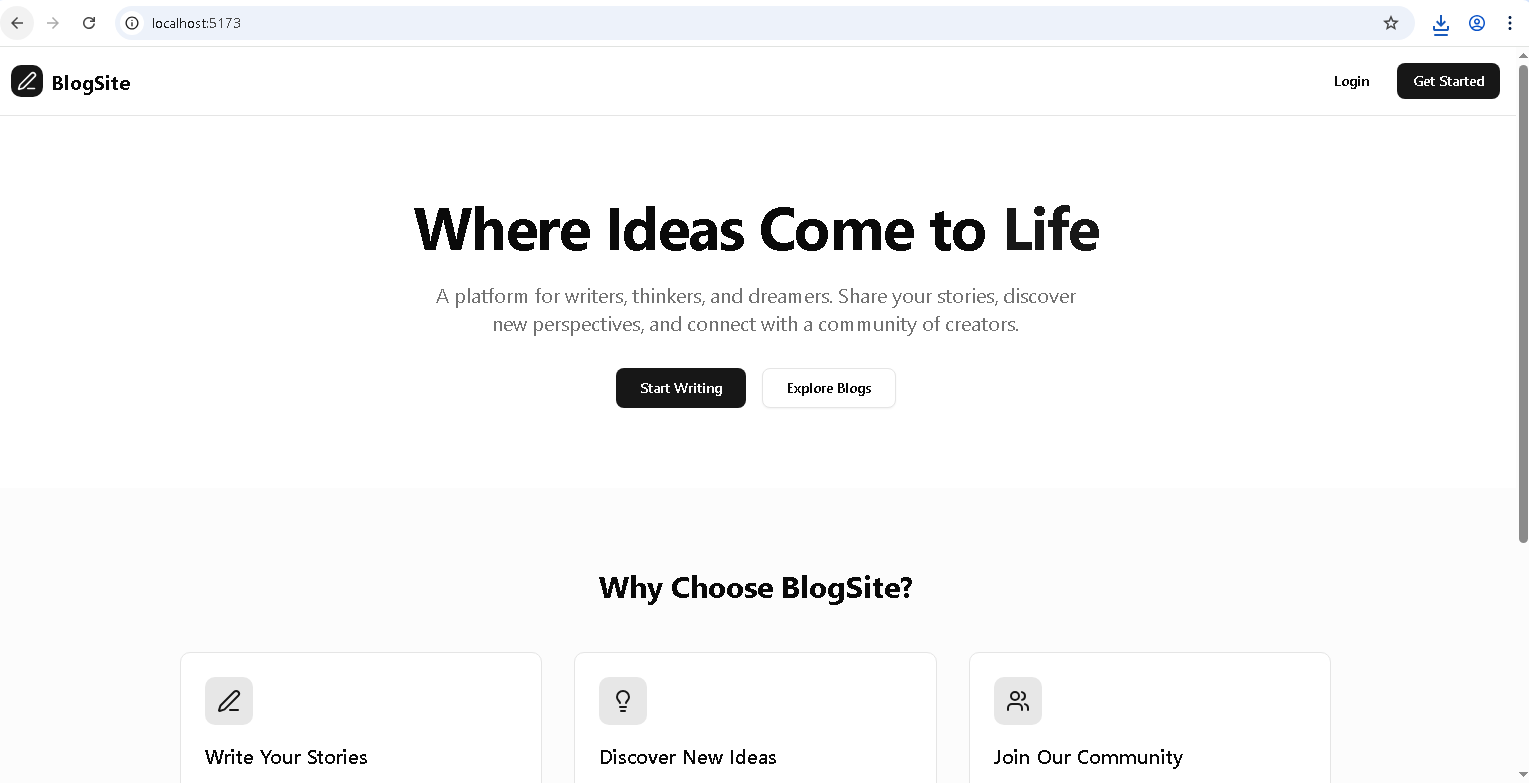
**Screenshot 2: Registration Page**



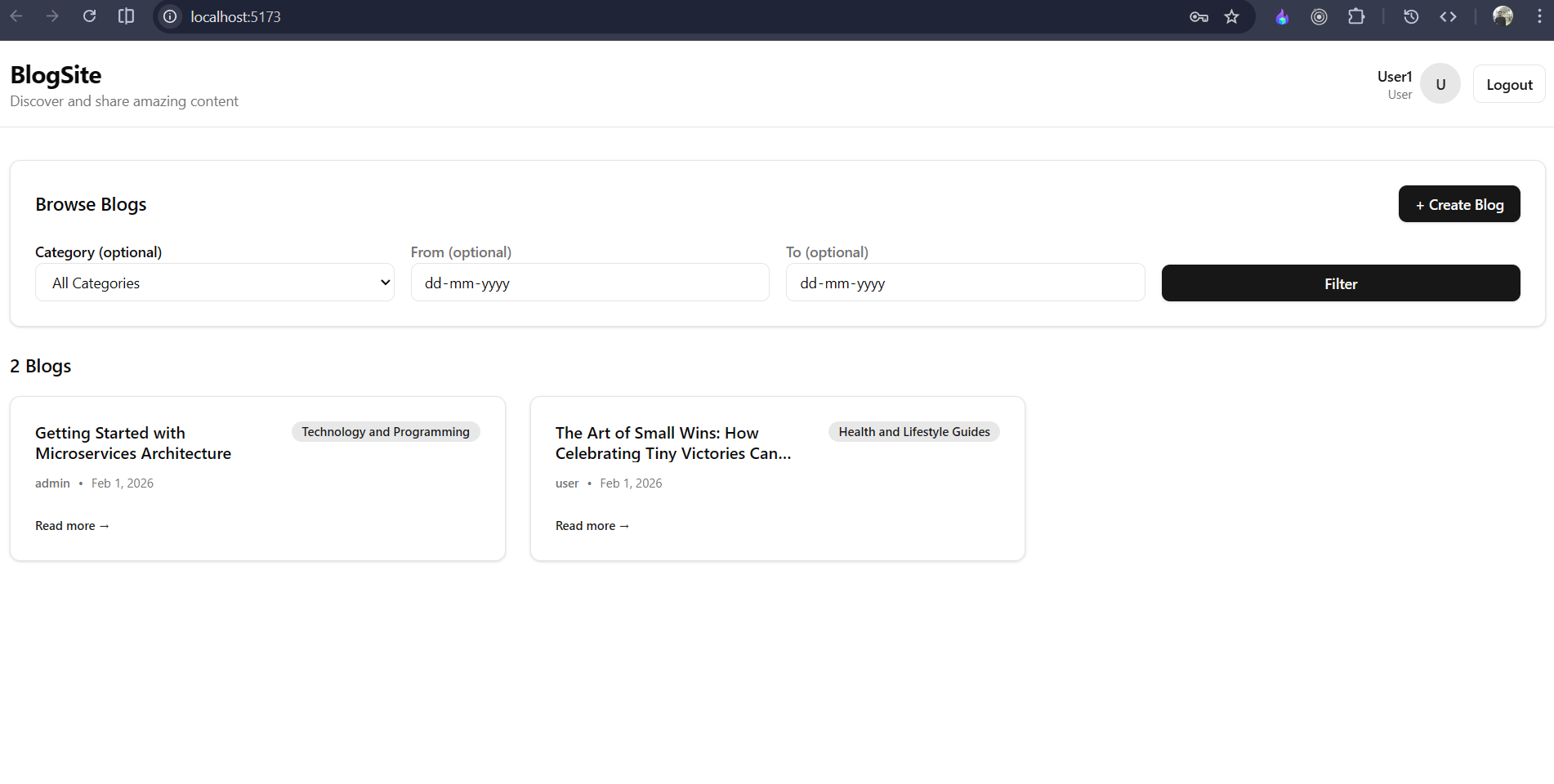
### 6.2 Blog Management

**Screenshot 3: Landing Page and Home Page - Blog List**

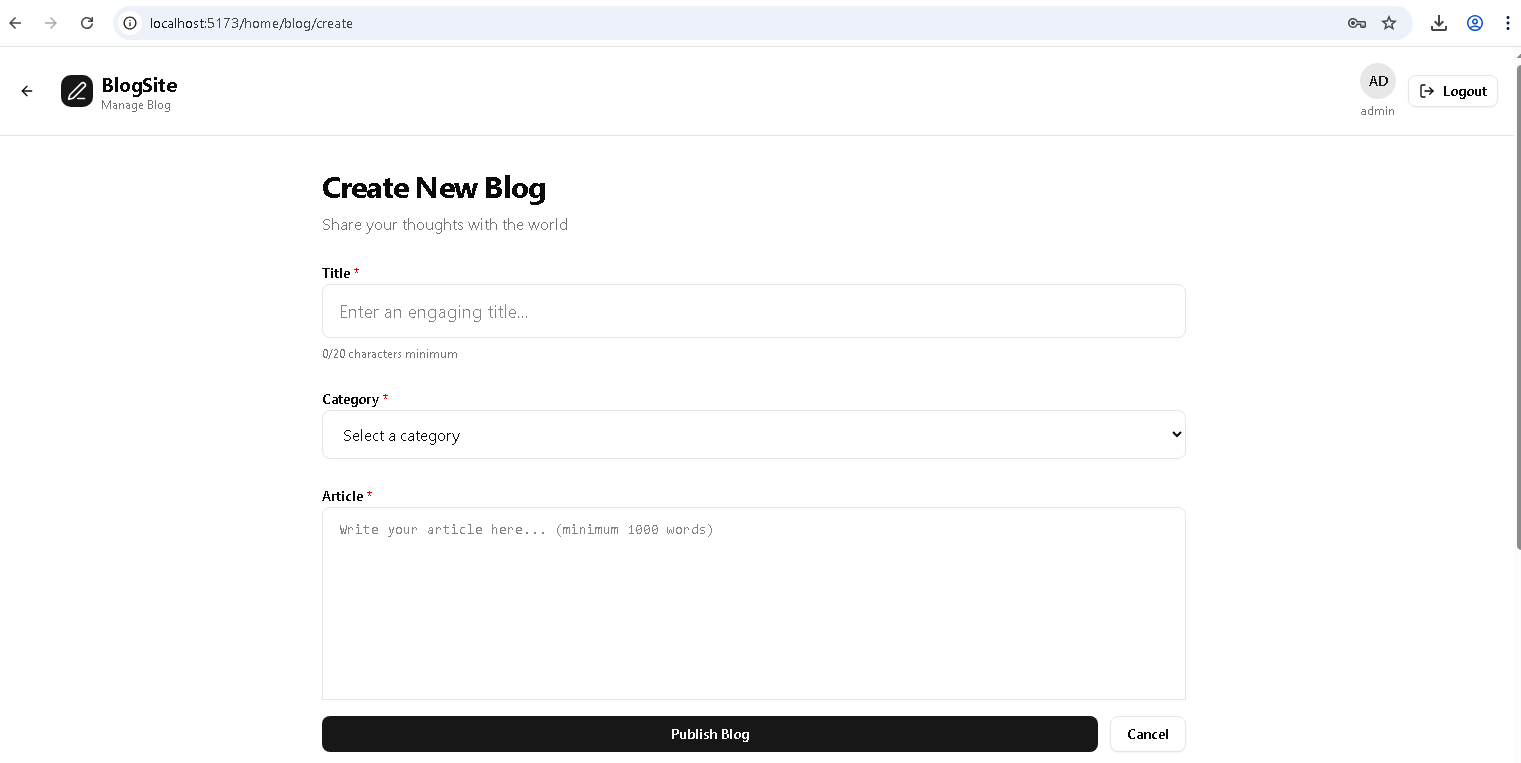
Landing Page



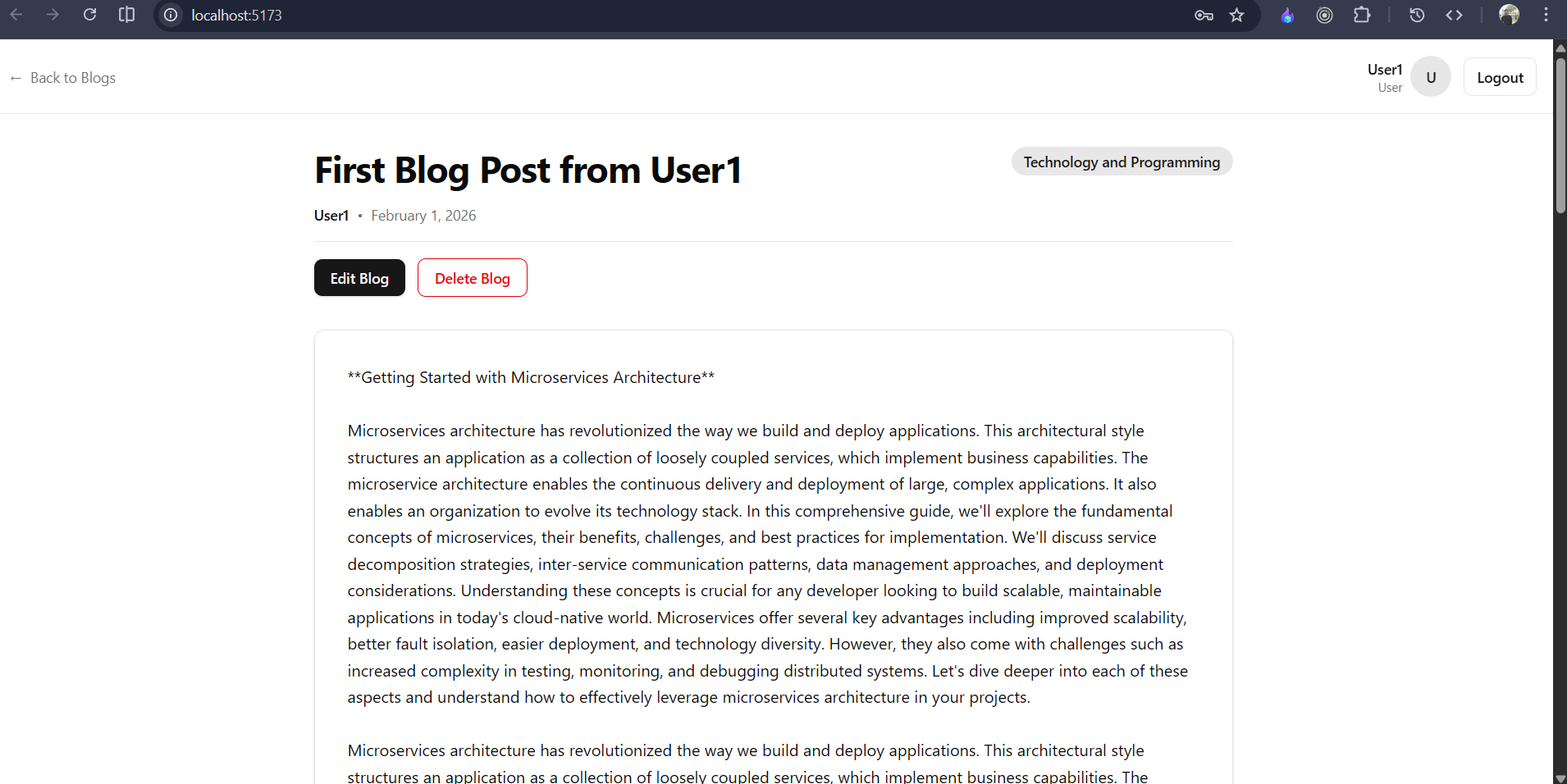
Home Page



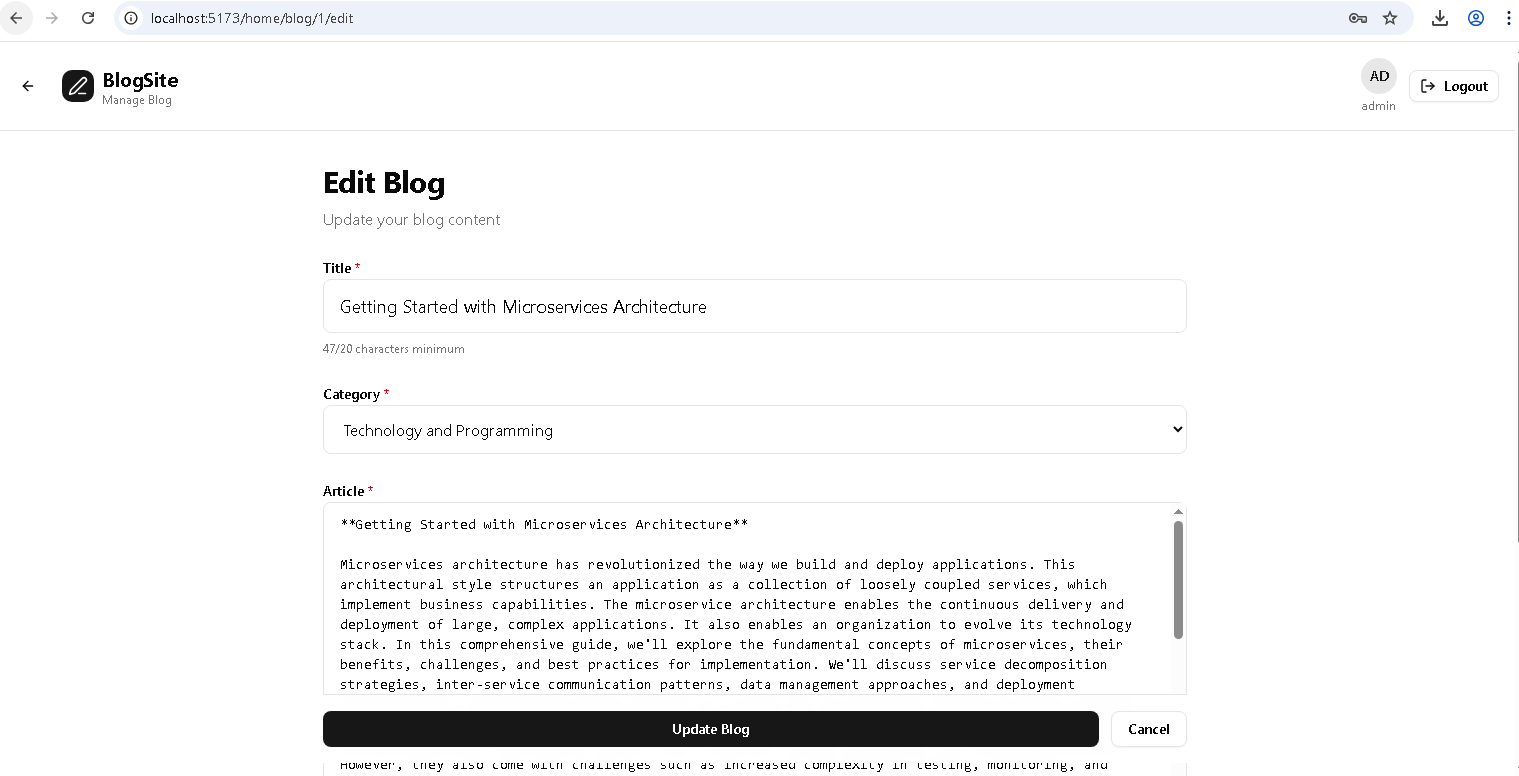
**Screenshot 4: Create New Blog**



**Screenshot 5: Blog Detail View**

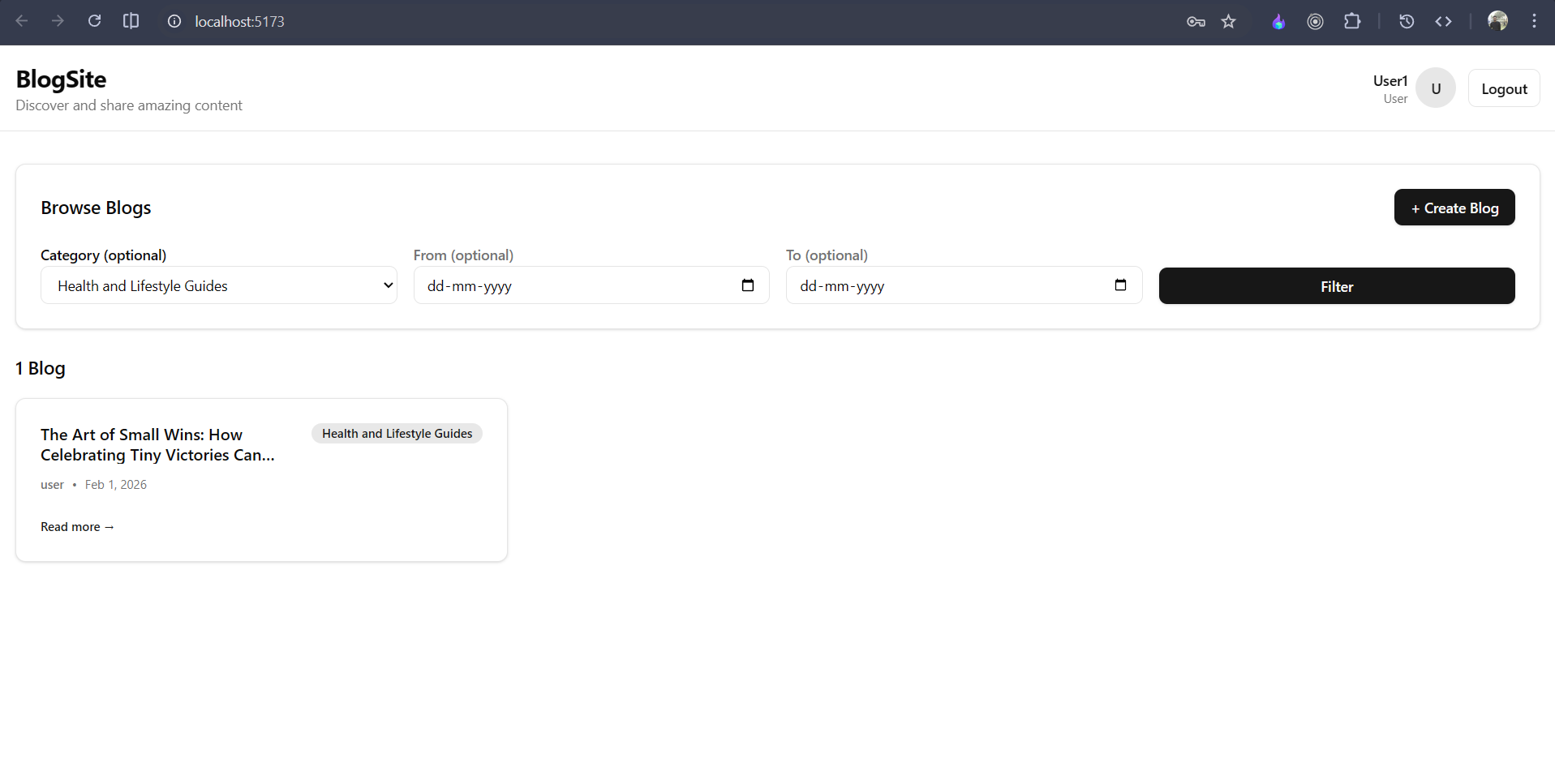
****

**Screenshot 6: Edit Blog**

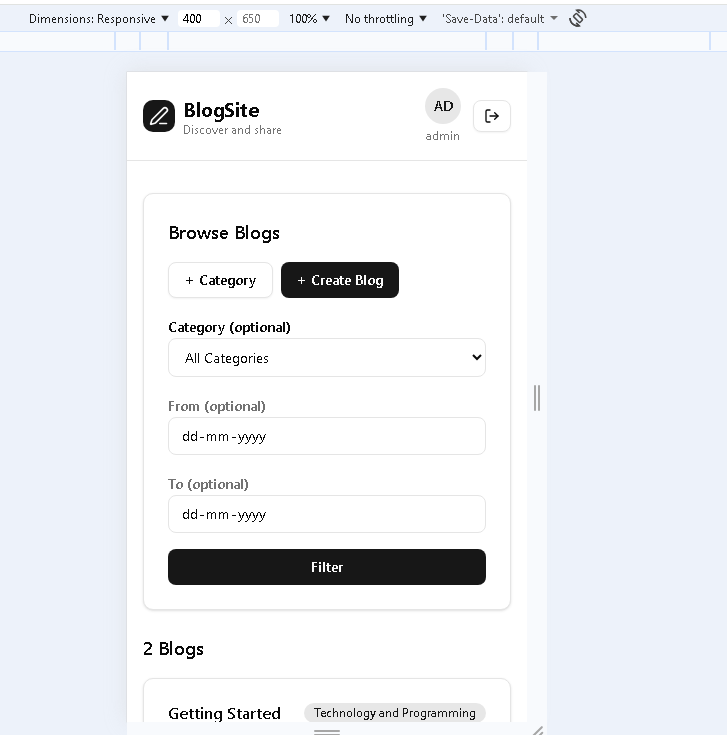


### 6.3 Category Filtering

**Screenshot 7: Category Filter**

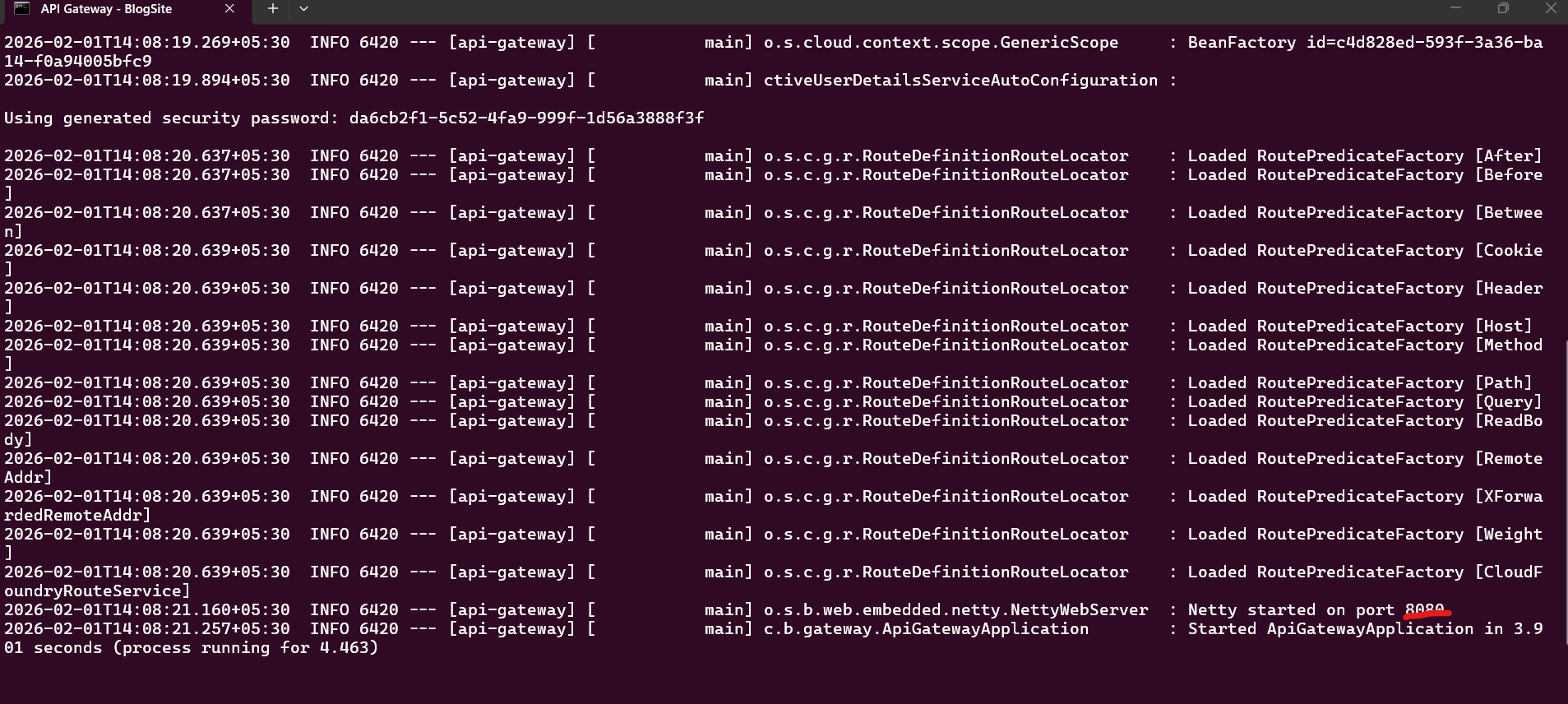


### 6.4 User Interface Features

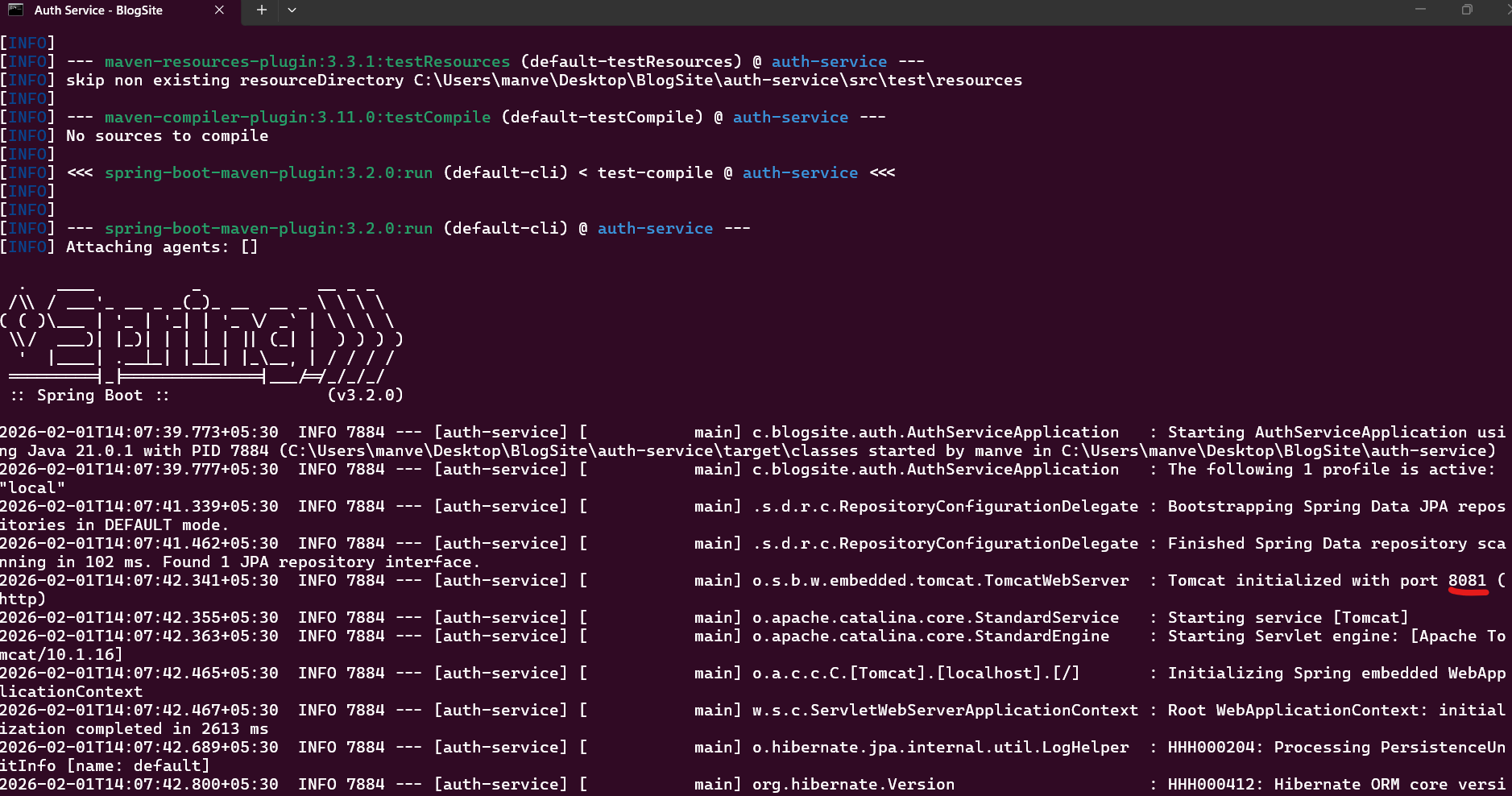
**Screenshot 8: Responsive Design - Mobile View** 

### 6.5 Backend Services

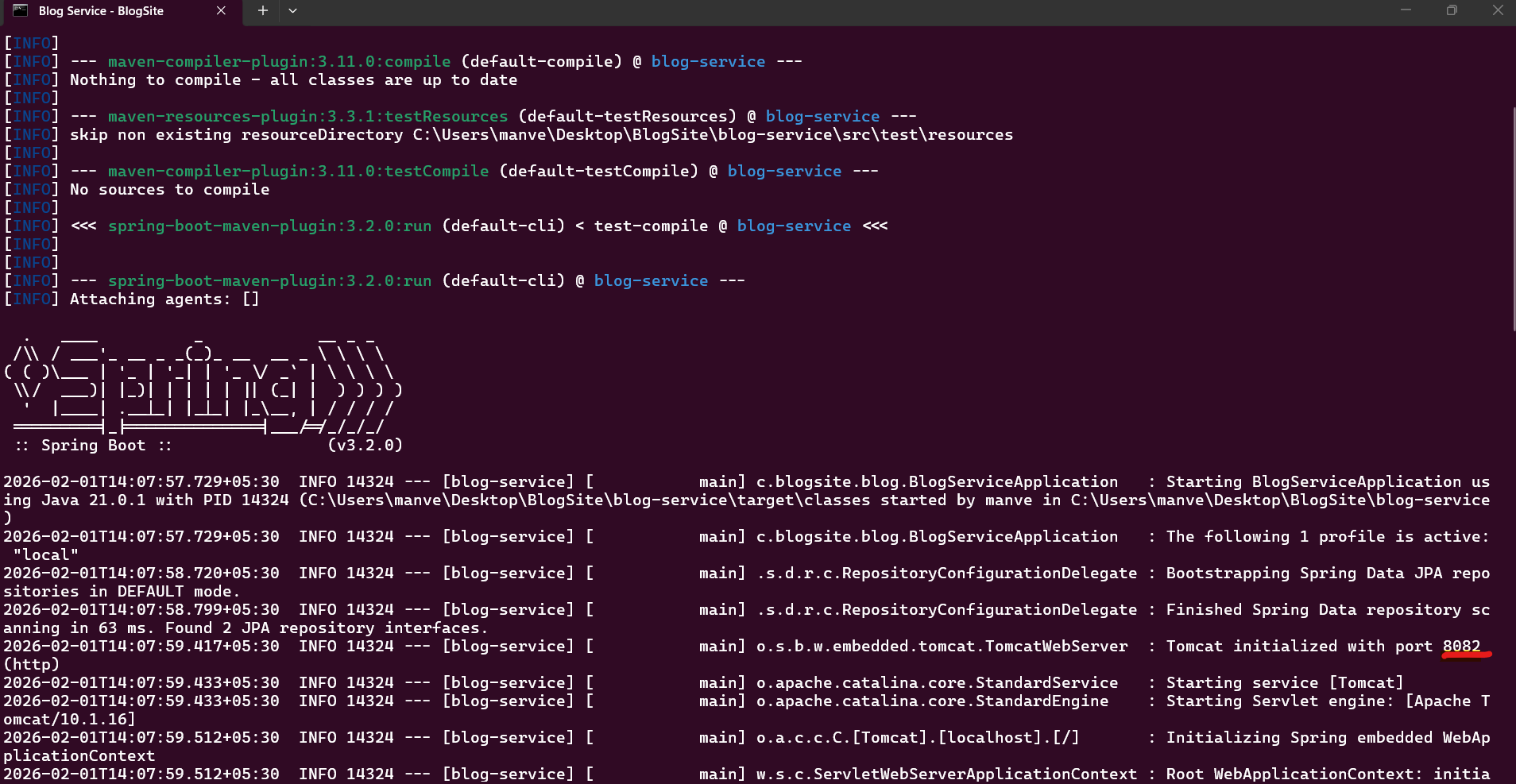
**Screenshot 9: API Gateway Console**



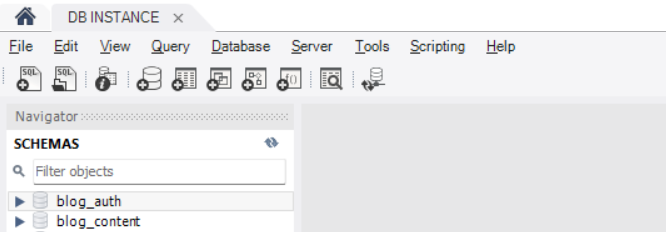
**Screenshot 10: Auth Service Console**

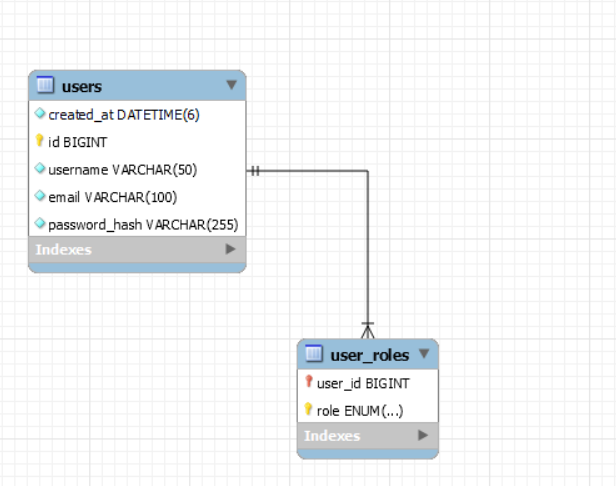
****

**Screenshot 11: Blog Service Console**



**Screenshot 12: MySQL Databases**







## 7. Setup and Execution

### 7.1 Prerequisites

* ✅ Java 21 or higher
* ✅ Maven 3.8+
* ✅ Node.js 18+
* ✅ MySQL 8.0
* ✅ Git

### 7.2 Database Setup

-- MySQL automatically creates these databases on first run  
CREATE DATABASE IF NOT EXISTS blog\_auth;  
CREATE DATABASE IF NOT EXISTS blog\_content;

### 7.3 One-Command Setup

The project includes an automated setup script:

# Run from project root  
setup-and-run.bat

This script:   
1. Installs frontend dependencies (npm install)   
2. Starts Auth Service (port 8081)   
3. Starts Blog Service (port 8082)   
4. Starts API Gateway (port 8080)   
5. Starts Frontend (port 5173)   
6. Opens browser automatically

### 7.4 Manual Setup (Alternative)

If you prefer manual setup:

# Terminal 1 - Auth Service  
cd auth-service  
mvn spring-boot:run -Dspring-boot.run.profiles=local  
  
# Terminal 2 - Blog Service  
cd blog-service  
mvn spring-boot:run -Dspring-boot.run.profiles=local  
  
# Terminal 3 - API Gateway  
cd api-gateway  
mvn spring-boot:run -Dspring-boot.run.profiles=local  
  
# Terminal 4 - Frontend  
cd frontend  
npm install  
npm run dev

### 7.5 Accessing the Application

* **Frontend:** http://localhost:5173
* **API Gateway:** http://localhost:8080
* **Auth Service:** http://localhost:8081
* **Blog Service:** http://localhost:8082

### 7.6 Default Credentials

For testing, use these pre-seeded accounts:

| Username | Password | Role | Purpose |
| --- | --- | --- | --- |
| admin | admin123 | ADMIN | Full access account |
| user | user123 | USER | Regular user account |

## 8. Project Structure

BlogSite/  
├── .github/  
│ └── copilot-instructions.md # Project documentation  
├── api-gateway/ # Spring Cloud Gateway  
│ ├── src/main/java/com/blogsite/gateway/  
│ │ ├── config/  
│ │ │ └── SecurityConfig.java # CORS & Security config  
│ │ ├── security/  
│ │ │ ├── JwtService.java # JWT validation  
│ │ │ └── GatewayAuthenticationFilter.java  
│ │ └── exception/  
│ │ └── GlobalErrorWebExceptionHandler.java  
│ └── pom.xml  
├── auth-service/ # Authentication microservice  
│ ├── src/main/java/com/blogsite/auth/  
│ │ ├── domain/  
│ │ │ └── User.java # User entity  
│ │ ├── repository/  
│ │ │ └── UserRepository.java  
│ │ ├── service/  
│ │ │ ├── JwtService.java # Token generation  
│ │ │ └── UserSeeder.java # Demo user seeding  
│ │ ├── web/  
│ │ │ ├── AuthController.java # Auth endpoints  
│ │ │ └── GlobalExceptionHandler.java  
│ │ └── config/  
│ │ └── SecurityConfig.java  
│ └── pom.xml  
├── blog-service/ # Blog management microservice  
│ ├── src/main/java/com/blogsite/blog/  
│ │ ├── domain/  
│ │ │ ├── Blog.java # Blog entity  
│ │ │ └── Category.java # Category entity  
│ │ ├── repository/  
│ │ │ ├── BlogRepository.java  
│ │ │ └── CategoryRepository.java  
│ │ ├── service/  
│ │ │ ├── BlogService.java  
│ │ │ ├── CategorySeeder.java # Demo categories  
│ │ │ └── BlogSeeder.java # Demo blogs  
│ │ ├── web/  
│ │ │ ├── BlogController.java # Blog CRUD endpoints  
│ │ │ └── GlobalExceptionHandler.java  
│ │ └── config/  
│ │ ├── SecurityConfig.java  
│ │ └── GatewayAuthenticationFilter.java  
│ └── pom.xml  
├── frontend/ # React frontend  
│ ├── src/  
│ │ ├── components/  
│ │ │ └── ui/ # Reusable UI components  
│ │ ├── pages/  
│ │ │ ├── LoginPage.jsx  
│ │ │ ├── HomePage.jsx  
│ │ │ └── BlogDetailPage.jsx  
│ │ ├── services/  
│ │ │ ├── apiClient.js # Axios with interceptors  
│ │ │ ├── authApi.js  
│ │ │ └── blogApi.js  
│ │ ├── hooks/  
│ │ │ └── useAuth.js # Authentication hook  
│ │ ├── constants/  
│ │ │ └── categories.js  
│ │ └── App.jsx  
│ ├── package.json  
│ └── vite.config.js  
├── docker-compose.yml # Docker deployment config  
├── setup-and-run.bat # One-command setup script  
├── stop-services.bat # Stop all services  
├── PRESENTATION.md # Detailed project documentation  
└── .gitignore

## 9. Security Features

### 9.1 Password Security

* Passwords encrypted using **BCrypt** hashing algorithm
* Salt automatically generated for each password
* Passwords never stored in plain text

### 9.2 JWT Security

* Tokens signed with **HS256** algorithm
* Secret key stored securely (not in repository)
* Token expiration enforced
* Separate access and refresh tokens for security

### 9.3 API Security

* All endpoints behind API Gateway
* JWT validation before routing requests
* Role-based access control (RBAC)
* CORS configured to prevent unauthorized access

### 9.4 Database Security

* Separate databases for each microservice
* Credentials configurable via environment variables
* Connection pooling for performance

### 9.5 Exception Handling

* Generic error messages to prevent information leakage
* Detailed logging for debugging (server-side only)
* No stack traces exposed to clients
* Standardized error response format

## 10. Project Zip Files

**📦 Zip File:**

****

**Zip Contents:**

BlogSite/  
├── All source code files  
├── Configuration files  
├── Setup scripts (setup-and-run.bat, stop-services.bat)  
└── .gitignore