Microblog Dashboard Flask App - Enhanced with User Profiles & Search

🆕 NEW FEATURES

- **User Profiles**: Individual user pages with analytics
- **Post Search**: Advanced search with pandas filtering
- **Read-Only Views**: Enhanced security with separate read/write database connections
- **Profile Analytics**: User activity analysis with pandas

Requirements (requirements.txt)

```
Flask==2.3.3
pandas==2.1.1
Werkzeug==2.3.7
```

Project Structure

```
microblog-dashboard/
│
├── app.py                 # Main Flask application (Enhanced)
├── requirements.txt       # Python dependencies
├── microblog.db           # SQLite database (auto-created)
└── templates/             # HTML templates
    ├── base.html          # Base template with enhanced navigation
    ├── login.html         # User login page
    ├── register.html      # User registration page
    ├── dashboard.html     # Main dashboard (4 user actions now)
    ├── posts.html         # View all posts (with profile links)
    ├── create_post.html   # Create new post
    ├── search_posts.html  # 🆕 Advanced post search
    ├── user_profile.html  # 🆕 User profile with analytics
    ├── admin_users.html   # Admin: User management
    └── admin_stats.html   # Admin: Advanced statistics
```

Installation & Quick Start

1. Setup Environment

```bash
# Create project directory
mkdir microblog-dashboard
cd microblog-dashboard

# Create virtual environment
python -m venv venv

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

2. Install Dependencies

```bash
pip install -r requirements.txt
```

3. Run Application

```bash
python app.py
```

4. Access Dashboard

- Open browser: `http://127.0.0.1:5000`
- **Default Admin Login:**
  - Username: `admin`
  - Password: `admin123`

🎯 Enhanced Dashboard Features

🔐 User Section (All Users)

- **Login/Logout System**: Secure authentication with password hashing
- **User Registration**: Create new accounts with validation
- **Create Posts**: Write microblogs (280 char limit)
- **View All Posts**: Browse community posts with profile links
- 🆕 **Search Posts**: Advanced pandas-powered search by content, user, or date
- 🆕 **User Profiles**: Personal profile pages with detailed analytics
- **Personal Stats**: Track your post count and activity

⚙️ Administration Section (Admin Only)

- **View Current Users**: Pandas-powered user table with post counts
- **Export Users**: Download user data as CSV using pandas
- **Export Database**: Complete SQLite database backup
- **Advanced Statistics**:
  - Interactive charts (Chart.js)
  - User activity analysis
  - Growth metrics
  - Most active users

🔧 Enhanced Technical Features

🔒 Security Improvements

Read-Only Database Connections

```python
def get_readonly_connection():
    """Get read-only database connection for safer queries"""
    conn = sqlite3.connect(f'file:{DATABASE}?mode=ro', uri=True)
    conn.row_factory = sqlite3.Row
    return conn

# Used for all data retrieval operations
conn = get_readonly_connection()
users = conn.execute('SELECT * FROM users').fetchall()
```

Error Handling & Input Validation

```python
try:
    conn = get_readonly_connection()
    # Safe database operations
except Exception as e:
    flash(f'Error: {str(e)}')
    return redirect(url_for('dashboard'))
```

🔍 Advanced Post Search

Pandas-Powered Search Engine

```python
@app.route('/search', methods=['GET'])
def search_posts():
    query = request.args.get('q', '').strip()
    search_type = request.args.get('type', 'content')

    # Load all posts into pandas DataFrame
    posts_df = pd.read_sql_query('''
        SELECT p.id, p.content, p.created_at, u.username, u.id as user_id
        FROM posts p JOIN users u ON p.user_id = u.id
        ORDER BY p.created_at DESC
    ''', conn)

    # Apply pandas filtering
    if search_type == 'content':
        mask = posts_df['content'].str.contains(query, case=False, na=False)
    elif search_type == 'user':
        mask = posts_df['username'].str.contains(query, case=False, na=False)
    elif search_type == 'date':
        posts_df['date_only'] = pd.to_datetime(posts_df['created_at']).dt.date.astype(str)
        mask = posts_df['date_only'].str.contains(query, na=False)

    filtered_posts = posts_df[mask].to_dict('records')
```

👤 User Profile Analytics

Profile Statistics with Pandas

```python
def user_profile(username):
    # Get user's posts
    posts_df = pd.read_sql_query('''
        SELECT * FROM posts WHERE user_id = ? ORDER BY created_at DESC
    ''', conn, params=(user['id'],))

    # Advanced analytics
    profile_stats = {
        'total_posts': len(posts_df),
        'recent_posts': len(posts_df[posts_df['created_at'] > recent_date]),
        'avg_post_length': round(posts_df['content'].str.len().mean(), 1),
        'most_active_day': posts_df['created_at'].dt.day_name().value_counts().index[0]
    }
```

🛡️ Enhanced Security Features

1. **Read-Only Database Access**: Separate connections for read operations
2. **Input Sanitization**: All user inputs properly validated
3. **Error Handling**: Comprehensive try-catch blocks
4. **Session Security**: Proper session management
5. **SQL Injection Prevention**: Parameterized queries only

📊 New Dashboard Layout

Homepage Dashboard (Enhanced)

1. **User Actions Grid (4 buttons)**:
    - Create Post
    - View All Posts
    - 🆕 **Search Posts**
    - 🆕 **My Profile**
2. **Enhanced Navigation**:
    - Dashboard | Posts | **Search** | **Profile** | Logout

🆕 User Profile Features

- **Profile Header**: Avatar, bio, member since
- **Activity Stats**: Total posts, recent activity, avg length, most active day
- **All User Posts**: Chronological post history
- **Profile Analytics**: Activity level badges, writing style analysis
- **Quick Actions**: Search user's posts, create post, navigation

🆕 Search Features

- **Multi-Type Search**: Content, Username, or Date
- **Pandas Filtering**: Advanced text matching and date filtering
- **Search Results**: Highlighted results with user profile links
- **Search Tips**: Built-in help for effective searching

🚀 Enhanced API Endpoints

Method	Endpoint	Description	Auth	New
GET	`/`	Enhanced dashboard	Login	✅
GET	`/search`	**Search posts**	Login	🆕
GET	`/profile/<username>`	**User profile**	Login	🆕
GET/POST	`/login`	User authentication	Public	✅
GET/POST	`/register`	User registration	Public	
POST	`/logout`	User logout	Login	
GET/POST	`/create_post`	Create new post	Login	
GET	`/posts`	View all posts (enhanced)	Login	✅
GET	`/admin/users`	User management	Admin	
GET	`/admin/stats`	Statistics dashboard	Admin	
GET	`/admin/export/users`	Export users CSV	Admin	
GET	`/admin/export/database`	Export database	Admin	
GET	`/api/users`	JSON user data	Admin	

📈 Pandas Usage Benefits (Enhanced)

1. **Advanced Search**: Complex text filtering and date operations
2. **Profile Analytics**: User behavior analysis and statistics
3. **Data Processing**: Complex queries with aggregations
4. **Export Functionality**: CSV generation with proper formatting
5. **Time Series Analysis**: Activity patterns and trends
6. **HTML Generation**: Formatted tables with styling
7. **API Responses**: Clean JSON output via `to_dict()`

🔮 Next Enhancement Ideas

User Experience

- **Follow System**: Follow other users
- **Post Reactions**: Likes, shares, comments
- **Trending Topics**: Popular hashtags and mentions
- **Real-time Updates**: WebSocket integration

Analytics & Data

- **Advanced Charts**: User engagement visualizations
- **Export Profiles**: Individual user data export
- **Bulk Operations**: Admin bulk user management
- **Data Import**: Excel/CSV user import
- **Machine Learning**: Content analysis and recommendations

This enhanced microblog now provides a complete social platform experience with robust user profiles, advanced search capabilities, and enterprise-level security practices—all powered by pandas for superior data processing and analytics.

Requirements (requirements.txt)

```
Flask==2.3.3
pandas==2.1.1
Werkzeug==2.3.7
```

Project Structure

```
microblog-dashboard/
│
├── app.py              # Main Flask application
├── requirements.txt       # Python dependencies
├── microblog.db         # SQLite database (auto-created)
└── templates/           # HTML templates
    ├── base.html          # Base template with navigation
    ├── login.html         # User login page
    ├── register.html      # User registration page
    ├── dashboard.html     # Main dashboard homepage
    ├── posts.html         # View all posts
    ├── create_post.html    # Create new post
    ├── admin_users.html    # Admin: User management
    └── admin_stats.html    # Admin: Advanced statistics
```

Installation & Quick Start

1. Setup Environment

```bash
# Create project directory
mkdir microblog-dashboard
cd microblog-dashboard

# Create virtual environment
python -m venv venv

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

2. Install Dependencies

```bash
pip install -r requirements.txt
```

3. Run Application

```bash
python app.py
```

4. Access Dashboard

- Open browser: `http://127.0.0.1:5000`
- **Default Admin Login:**
  - Username: `admin`
  - Password: `admin123`

🎯 Dashboard Features Overview

🔐 User Section (All Users)

- **Login/Logout System**: Secure authentication with password hashing
- **User Registration**: Create new accounts with validation
- **Create Posts**: Write microblogs (280 char limit, like Twitter)
- **View Posts**: Browse all community posts
- **Personal Stats**: Track your post count

⚙️ Administration Section (Admin Only)

- **View Current Users**: Pandas-powered user table with post counts
- **Export Users**: Download user data as CSV using pandas
- **Export Database**: Complete SQLite database backup
- **Advanced Statistics**:
  - Interactive charts (Chart.js)
  - User activity analysis
  - Growth metrics
  - Most active users

🔧 Key Technical Features

Pandas Integration Examples

1. User Management Table

```python
# Query with JOIN and aggregation
query = '''
  SELECT u.id, u.username, u.email, u.created_at, u.is_admin,
      COUNT(p.id) as post_count
  FROM users u
  LEFT JOIN posts p ON u.id = p.user_id
  GROUP BY u.id, u.username, u.email, u.created_at, u.is_admin
  ORDER BY u.created_at DESC
'''

df = pd.read_sql_query(query, conn)
df['created_at'] = pd.to_datetime(df['created_at']).dt.strftime('%Y-%m-%d %H:%M')
users_table = df.to_html(classes='table table-striped')
```

2. CSV Export Functionality

```python
@app.route('/admin/export/users')
def export_users():
  df = pd.read_sql_query(complex_query, conn)

  # Export to CSV in memory
  output = io.StringIO()
  df.to_csv(output, index=False)

  # Return as downloadable file
  response = make_response(output.getvalue())
  response.headers["Content-Disposition"] = "attachment; filename=users_export.csv"
  return response
```

3. Statistics & Analytics

```python
# Time series analysis
users_df['created_at'] = pd.to_datetime(users_df['created_at'])
users_by_month = users_df.groupby(users_df['created_at'].dt.to_period('M')).size()

# Activity analysis
posts_per_user = posts_df['username'].value_counts()
most_active_user = posts_per_user.index[0]

# Recent activity filtering
recent_posts = posts_df[posts_df['created_at'] >
  (pd.Timestamp.now() - pd.Timedelta(days=7))]
```

Security Features

- Password hashing with Werkzeug
- Session management
- Login required decorators
- Admin-only route protection
- CSRF protection via Flask sessions

Database Design

```sql
-- Users table
CREATE TABLE users (
  id INTEGER PRIMARY KEY AUTOINCREMENT,
  username TEXT UNIQUE NOT NULL,
  email TEXT UNIQUE NOT NULL,
  password_hash TEXT NOT NULL,
  is_admin BOOLEAN DEFAULT 0,
  created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);

-- Posts table with foreign key
CREATE TABLE posts (
  id INTEGER PRIMARY KEY AUTOINCREMENT,
  user_id INTEGER NOT NULL,
  content TEXT NOT NULL,
  created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
  FOREIGN KEY (user_id) REFERENCES users (id)
);
```

📊 Dashboard Sections Explained

Homepage Dashboard Layout

1. **Left Column (User Actions)**:
    - Create new posts
    - View all posts
    - Personal post counter
    - Recent activity feed
2. **Right Column (Context)**:
    - Admin panel (if admin user)
    - Account information
    - Quick statistics

Admin Panel Features

- **User Management**: Sortable table with post counts
- **Data Export**: CSV and database backup
- **Analytics Dashboard**:
    - Real-time charts
    - User growth metrics
    - Activity trends
    - Top contributors

🚀 API Endpoints

Method	Endpoint	Description	Auth Required
GET	`/`	Main dashboard	Login
GET/POST	`/login`	User authentication	Public
GET/POST	`/register`	User registration	Public
POST	`/logout`	User logout	Login
GET/POST	`/create_post`	Create new post	Login
GET	`/posts`	View all posts	Login
GET	`/admin/users`	User management	Admin
GET	`/admin/stats`	Statistics dashboard	Admin
GET	`/admin/export/users`	Export users CSV	Admin
GET	`/admin/export/database`	Export database	Admin
GET	`/api/users`	JSON user data	Admin

📈 Pandas Usage Benefits

1. **Data Processing**: Complex queries with aggregations
2. **Export Functionality**: CSV generation with proper formatting
3. **Analytics**: Time series analysis and statistics
4. **HTML Generation**: Formatted tables with styling
5. **API Responses**: Clean JSON output via `to_dict()`

🔮 Enhancement Ideas

Immediate Improvements

- **User Profiles**: Individual user pages with activity
- **Post Search**: Pandas-powered search and filtering
- **Data Visualization**: Charts for user engagement
- **Bulk Operations**: Admin bulk user management

Advanced Features

- **Real-time Dashboard**: WebSocket integration
- **Machine Learning**: Content analysis with pandas/scikit-learn
- **Advanced Analytics**: User behavior patterns
- **API Extensions**: RESTful API with pandas processing
- **Data Import**: Excel/CSV user import functionality

This dashboard-focused microblog demonstrates practical pandas integration in web applications, particularly for admin interfaces, data export, and analytics - common requirements in real-world applications.