# **Blog Post Application Documentation**

## **Introduction**

This document provides a comprehensive explanation of the Blog Post Application, its functionality, and the underlying SQL code that supports its database schema. The project is a blogging platform that enables users to create, manage, and interact with posts, categories, tags, and profiles. This guide explains how the application was designed and implemented, covering both the front-end and back-end operations.

## **Application Features**

### **User Management**

#### **Sign-Up and Sign-In**

* **Process**:  
  + Users can sign up with a unique email and username.
  + Passwords are securely hashed before being stored in the database.
  + JWT-based authentication is implemented for secure session management.
* **Database Structure**:  
  + The User table stores user information, including their email, username, password, and authToken.

#### **Profile Management**

* Users can update their profile, including uploading avatars and modifying their details.
* Related database fields include avatar, name, and updatedAt in the User table.

### **Post Management**

#### **Creating Posts**

* Users can create posts with a title, content, and optional tags and categories.
* Posts default to an unpublished state and can be published later.

**SQL Example**:  
 INSERT INTO posts (title, content, authorId, published, createdAt)

VALUES ('My First Post', 'This is the content of the post.', 'user-id', false, NOW());

#### **Viewing and Incrementing View Count**

* Posts have a viewCount column that tracks the number of views.

A stored procedure increments this count:  
 CREATE OR REPLACE FUNCTION increment\_view\_count(postId UUID) RETURNS VOID AS $$

BEGIN

UPDATE posts SET viewCount = viewCount + 1 WHERE id = postId;

END;

$$ LANGUAGE plpgsql;

#### **Deleting Posts**

* Users can delete their posts. Deletion cascades remove related comments and tags.

**SQL Example**:  
 DELETE FROM posts WHERE id = 'post-id';

### **Tag and Category Management**

#### **Adding Tags and Categories**

* Tags and categories help organize posts.

**SQL Example**:  
 INSERT INTO tags (name) VALUES ('Tech');

INSERT INTO categories (name) VALUES ('Programming');

#### **Associating Tags and Categories**

* The post\_tags junction table connects tags to posts.

**SQL Example**:  
 INSERT INTO post\_tags (post\_id, tag\_id) VALUES ('post-id', 'tag-id');

#### **Removing Tags and Categories**

* Tags and categories can be removed. Deletion cascades ensure consistency.

**SQL Example**:  
 DELETE FROM tags WHERE id = 'tag-id';

### **Searching and Filtering**

* Users can search for posts based on keywords in titles or content.
* Posts can also be filtered by category or tag.

**SQL Example**:  
 SELECT \* FROM posts WHERE title ILIKE '%keyword%' OR content ILIKE '%keyword%';

### **Settings and Customizations**

* Users can update their account settings, including email, username, and password.
* Password changes are securely handled by verifying the old password and hashing the new one.

## **Database Schema and SQL Explanation**

### **User Table**

CREATE TABLE users (

id UUID PRIMARY KEY DEFAULT gen\_random\_uuid(),

name TEXT,

username TEXT UNIQUE,

email TEXT UNIQUE,

password TEXT,

verifyToken TEXT,

authToken TEXT,

avatar TEXT,

createdAt TIMESTAMP DEFAULT NOW(),

updatedAt TIMESTAMP DEFAULT NOW()

);

* **Purpose**: Stores user details and manages authentication.
* **Key Fields**:
  + id: Unique identifier for each user.
  + authToken: Used for session management.

### **Post Table**

CREATE TABLE posts (

id UUID PRIMARY KEY DEFAULT gen\_random\_uuid(),

title TEXT,

content TEXT,

published BOOLEAN DEFAULT FALSE,

authorId UUID REFERENCES users(id),

viewCount INT DEFAULT 0,

category\_id UUID REFERENCES categories(id),

createdAt TIMESTAMP DEFAULT NOW(),

updatedAt TIMESTAMP DEFAULT NOW()

);

* **Purpose**: Stores blog posts with metadata such as view count and category.

### **Tag Table**

CREATE TABLE tags (

id UUID PRIMARY KEY DEFAULT gen\_random\_uuid(),

name TEXT UNIQUE

);

* **Purpose**: Stores tags for categorizing posts.

### **Category Table**

CREATE TABLE categories (

id UUID PRIMARY KEY DEFAULT gen\_random\_uuid(),

name TEXT UNIQUE

);

* **Purpose**: Stores categories for organizing posts.

### **Junction Table for Tags and Posts**

CREATE TABLE post\_tags (

post\_id UUID REFERENCES posts(id) ON DELETE CASCADE,

tag\_id UUID REFERENCES tags(id) ON DELETE CASCADE,

PRIMARY KEY (post\_id, tag\_id)

);

* **Purpose**: Links posts to tags for many-to-many relationships.

## **Summary**

The Blog Post Application is designed to provide an efficient and user-friendly platform for creating, managing, and discovering content. The relational database schema ensures data consistency and supports complex queries for features like tagging, categorization, and view tracking. With robust authentication and CRUD operations, the application delivers a seamless blogging experience.