CSU CCIS Blog Application

## Complete Project Documentation

Colorado State University

College of Computing and Information Sciences

Generated: May 23, 2025

# Table of Contents

1. 1. Project Overview
2. 2. Entity-Relationship Diagram
3. 3. Database Schema
4. 4. API Specification
5. 5. Authentication Flow
6. 6. Setup Guide
7. 7. Usage Guide
8. 8. Code Structure
9. 9. Technologies Used

# Project Overview

## CSU CCIS Blog - Project Documentation

### Project Overview

#### Purpose

The CSU CCIS Blog is a web application designed for the College of Computing and Information Sciences community to share knowledge, experiences, and insights. This platform enables students, faculty, and staff to create, share, and discuss blog posts in a collaborative environment.

#### Chosen Tech Stack

* \*\*Backend Framework\*\*: Flask (Python)
* \*\*Database\*\*: PostgreSQL with SQLAlchemy ORM
* \*\*Frontend\*\*: HTML5, CSS3, JavaScript (Vanilla), Bootstrap 5
* \*\*Authentication\*\*: Flask-Login for session-based authentication
* \*\*Forms\*\*: Flask-WTF for form handling and validation
* \*\*Web Server\*\*: Gunicorn (Production)
* \*\*Deployment\*\*: Replit Platform

#### High-Level Architecture

┌─────────────────┐ ┌─────────────────┐ ┌─────────────────┐

│ Frontend │ │ Backend │ │ Database │

│ (HTML/CSS/JS) │◄──►│ (Flask) │◄──►│ (PostgreSQL) │

│ Bootstrap │ │ SQLAlchemy │ │ │

└─────────────────┘ └─────────────────┘ └─────────────────┘

│ │ │

│ ┌─────────────────┐ │

└─────────────►│ Authentication │◄─────────────┘

│ (Flask-Login) │

└─────────────────┘

### Features

#### Core Functionality

1. \*\*User Authentication\*\*: Registration, login, logout, session management
2. \*\*Blog Post Management\*\*: Create, read, update, delete (CRUD) operations
3. \*\*Comment System\*\*: Users can comment on posts
4. \*\*User Dashboard\*\*: Personal post management interface
5. \*\*Responsive Design\*\*: Mobile-friendly interface with CSU CCIS branding

#### Authentication Features

* Secure password hashing using Werkzeug
* Session-based authentication with Flask-Login
* User registration with email validation
* Login/logout functionality
* Protected routes requiring authentication

#### CRUD Operations

1. \*\*Posts\*\*: Full CRUD functionality for blog posts
2. \*\*Comments\*\*: Create and delete comments on posts
3. \*\*User Management\*\*: User registration and profile management

### Technical Requirements Met

✅ \*\*Frontend\*\*: Responsive UI with Bootstrap, client-side JavaScript for interactivity

✅ \*\*Backend\*\*: RESTful routing with Flask, business logic implementation

✅ \*\*Database\*\*: Proper schema design with relationships and data persistence

✅ \*\*Authentication\*\*: Complete signup, signin, signout workflow

✅ \*\*Two CRUD Modules\*\*: Posts and Comments with full functionality

### Project Structure

csu-ccis-blog/

├── app.py # Flask application configuration

├── main.py # Application entry point

├── models.py # Database models

├── routes.py # Application routes

├── forms.py # WTForms form definitions

├── utils.py # Utility functions

├── templates/ # Jinja2 templates

│ ├── base.html

│ ├── index.html

│ ├── login.html

│ ├── register.html

│ ├── dashboard.html

│ ├── create\_post.html

│ ├── edit\_post.html

│ ├── post\_detail.html

│ └── error.html

├── static/ # Static assets

│ ├── css/

│ │ └── style.css

│ ├── js/

│ │ ├── main.js

│ │ ├── posts.js

│ │ └── comments.js

│ └── images/

│ └── csu-ccis-logo.svg

└── docs/ # Documentation

├── ERD.md

├── API\_SPEC.md

├── SETUP\_GUIDE.md

└── USAGE\_GUIDE.md

### Development Team

* \*\*Framework\*\*: Individual/Team Project
* \*\*Timeline\*\*: 21 calendar days
* \*\*Institution\*\*: Colorado State University, College of Computing and Information Sciences

### License

Educational use for CSU CCIS coursework.

# Entity-Relationship Diagram

## Entity-Relationship Diagram (ERD)

### Database Schema Overview

The CSU CCIS Blog uses a relational database design with three main entities: Users, Posts, and Comments.

┌─────────────────┐ 1 ∞ ┌─────────────────┐ 1 ∞ ┌─────────────────┐

│ USER │◄──────────│ POST │◄──────────│ COMMENT │

├─────────────────┤ ├─────────────────┤ ├─────────────────┤

│ id (PK) │ │ id (PK) │ │ id (PK) │

│ username │ │ title │ │ content │

│ email │ │ content │ │ created\_at │

│ password\_hash │ │ created\_at │ │ updated\_at │

│ created\_at │ │ updated\_at │ │ user\_id (FK) │

└─────────────────┘ │ user\_id (FK) │ │ post\_id (FK) │

└─────────────────┘ └─────────────────┘

### Relationships

1. \*\*User → Posts\*\*: One-to-Many

* One user can create multiple posts
* Each post belongs to exactly one user
* Foreign Key: `post.user\_id` references `user.id`

1. \*\*Post → Comments\*\*: One-to-Many

* One post can have multiple comments
* Each comment belongs to exactly one post
* Foreign Key: `comment.post\_id` references `post.id`

1. \*\*User → Comments\*\*: One-to-Many

* One user can create multiple comments
* Each comment belongs to exactly one user
* Foreign Key: `comment.user\_id` references `user.id`

### Database Constraints

* \*\*Primary Keys\*\*: Auto-incrementing integers for all entities
* \*\*Unique Constraints\*\*: username and email must be unique
* \*\*NOT NULL Constraints\*\*: All required fields must have values
* \*\*Cascade Deletes\*\*: When a user is deleted, their posts and comments are also deleted
* \*\*Foreign Key Constraints\*\*: Ensure referential integrity between tables

# Database Schema

## Database Schema Specifications

### Table Definitions

#### User Table

CREATE TABLE user (

id INTEGER PRIMARY KEY AUTOINCREMENT,

username VARCHAR(64) NOT NULL UNIQUE,

email VARCHAR(120) NOT NULL UNIQUE,

password\_hash VARCHAR(256) NOT NULL,

created\_at DATETIME DEFAULT CURRENT\_TIMESTAMP

);

\*\*Fields:\*\*

* `id`: Primary key, auto-incrementing integer
* `username`: Unique username, 3-64 characters
* `email`: Unique email address, valid email format
* `password\_hash`: Hashed password using Werkzeug security
* `created\_at`: Timestamp of account creation

#### Post Table

CREATE TABLE post (

id INTEGER PRIMARY KEY AUTOINCREMENT,

title VARCHAR(128) NOT NULL,

content TEXT NOT NULL,

created\_at DATETIME DEFAULT CURRENT\_TIMESTAMP,

updated\_at DATETIME DEFAULT CURRENT\_TIMESTAMP ON UPDATE CURRENT\_TIMESTAMP,

user\_id INTEGER NOT NULL,

FOREIGN KEY (user\_id) REFERENCES user(id) ON DELETE CASCADE

);

\*\*Fields:\*\*

* `id`: Primary key, auto-incrementing integer
* `title`: Post title, 3-128 characters
* `content`: Post content, unlimited text
* `created\_at`: Timestamp of post creation
* `updated\_at`: Timestamp of last modification
* `user\_id`: Foreign key referencing user.id

#### Comment Table

CREATE TABLE comment (

id INTEGER PRIMARY KEY AUTOINCREMENT,

content TEXT NOT NULL,

created\_at DATETIME DEFAULT CURRENT\_TIMESTAMP,

updated\_at DATETIME DEFAULT CURRENT\_TIMESTAMP ON UPDATE CURRENT\_TIMESTAMP,

user\_id INTEGER NOT NULL,

post\_id INTEGER NOT NULL,

FOREIGN KEY (user\_id) REFERENCES user(id) ON DELETE CASCADE,

FOREIGN KEY (post\_id) REFERENCES post(id) ON DELETE CASCADE

);

\*\*Fields:\*\*

* `id`: Primary key, auto-incrementing integer
* `content`: Comment content, unlimited text
* `created\_at`: Timestamp of comment creation
* `updated\_at`: Timestamp of last modification
* `user\_id`: Foreign key referencing user.id
* `post\_id`: Foreign key referencing post.id

### Relationships and Constraints

#### Primary Keys

* All tables use auto-incrementing integer primary keys
* Ensures unique identification of each record

#### Foreign Key Relationships

1. `post.user\_id` → `user.id` (CASCADE DELETE)
2. `comment.user\_id` → `user.id` (CASCADE DELETE)
3. `comment.post\_id` → `post.id` (CASCADE DELETE)

#### Unique Constraints

* `user.username`: Must be unique across all users
* `user.email`: Must be unique across all users

#### Data Validation

* Username: 3-64 characters, alphanumeric
* Email: Valid email format required
* Password: Minimum 6 characters (hashed)
* Post title: 3-128 characters
* Content fields: Required, cannot be empty

#### Indexes (Recommended)

CREATE INDEX idx\_post\_user\_id ON post(user\_id);

CREATE INDEX idx\_comment\_post\_id ON comment(post\_id);

CREATE INDEX idx\_comment\_user\_id ON comment(user\_id);

CREATE INDEX idx\_post\_created\_at ON post(created\_at);

# API Specification

## API Specification

### Authentication Endpoints

#### Register User

* \*\*Method\*\*: POST
* \*\*URL\*\*: `/register`
* \*\*Description\*\*: Create a new user account
* \*\*Request Body\*\*: Form data

username: string (3-64 characters)

email: string (valid email)

password: string (min 6 characters)

confirm\_password: string (must match password)

* \*\*Response\*\*: Redirect to login page on success
* \*\*Errors\*\*: 400 if validation fails

#### Login User

* \*\*Method\*\*: POST
* \*\*URL\*\*: `/login`
* \*\*Description\*\*: Authenticate user and create session
* \*\*Request Body\*\*: Form data

email: string

password: string

* \*\*Response\*\*: Redirect to dashboard on success
* \*\*Errors\*\*: 401 if credentials invalid

#### Logout User

* \*\*Method\*\*: GET
* \*\*URL\*\*: `/logout`
* \*\*Description\*\*: End user session
* \*\*Authentication\*\*: Required
* \*\*Response\*\*: Redirect to home page

### Post Management Endpoints

#### Get All Posts

* \*\*Method\*\*: GET
* \*\*URL\*\*: `/`
* \*\*Description\*\*: Display paginated list of all posts
* \*\*Query Parameters\*\*:

page: integer (default: 1)

* \*\*Response\*\*: HTML page with post list

#### View Post Detail

* \*\*Method\*\*: GET
* \*\*URL\*\*: `/post/<int:post\_id>`
* \*\*Description\*\*: Display single post with comments
* \*\*Response\*\*: HTML page with post details and comments

#### Create Post

* \*\*Method\*\*: GET/POST
* \*\*URL\*\*: `/post/new`
* \*\*Description\*\*: Create new blog post
* \*\*Authentication\*\*: Required
* \*\*Request Body\*\* (POST): Form data

title: string (3-128 characters)

content: string (required)

* \*\*Response\*\*: Redirect to dashboard on success

#### Edit Post

* \*\*Method\*\*: GET/POST
* \*\*URL\*\*: `/post/<int:post\_id>/edit`
* \*\*Description\*\*: Edit existing post
* \*\*Authentication\*\*: Required (must be post author)
* \*\*Request Body\*\* (POST): Form data

title: string (3-128 characters)

content: string (required)

* \*\*Response\*\*: Redirect to post detail on success

#### Delete Post

* \*\*Method\*\*: POST
* \*\*URL\*\*: `/post/<int:post\_id>/delete`
* \*\*Description\*\*: Delete existing post
* \*\*Authentication\*\*: Required (must be post author)
* \*\*Response\*\*: JSON

{

"success": true,

"message": "Post deleted successfully"

}

### Comment Management Endpoints

#### Add Comment

* \*\*Method\*\*: POST
* \*\*URL\*\*: `/post/<int:post\_id>/comment`
* \*\*Description\*\*: Add comment to post
* \*\*Authentication\*\*: Required
* \*\*Request Body\*\*: Form data

content: string (required)

* \*\*Response\*\*: Redirect to post detail page

#### Delete Comment

* \*\*Method\*\*: POST
* \*\*URL\*\*: `/comment/<int:comment\_id>/delete`
* \*\*Description\*\*: Delete comment
* \*\*Authentication\*\*: Required (must be comment author or post author)
* \*\*Response\*\*: JSON

{

"success": true,

"message": "Comment deleted successfully"

}

### Dashboard Endpoints

#### User Dashboard

* \*\*Method\*\*: GET
* \*\*URL\*\*: `/dashboard`
* \*\*Description\*\*: Display user's posts management interface
* \*\*Authentication\*\*: Required
* \*\*Response\*\*: HTML page with user's posts

### API Endpoints (JSON)

#### Get Posts API

* \*\*Method\*\*: GET
* \*\*URL\*\*: `/api/posts`
* \*\*Description\*\*: Get all posts as JSON
* \*\*Response\*\*: JSON array of posts

[

{

"id": 1,

"title": "Post Title",

"content": "Post content...",

"created\_at": "2025-05-23T18:00:00",

"updated\_at": "2025-05-23T18:00:00",

"author": "username",

"user\_id": 1,

"comment\_count": 5

}

]

#### Get Single Post API

* \*\*Method\*\*: GET
* \*\*URL\*\*: `/api/posts/<int:post\_id>`
* \*\*Description\*\*: Get specific post as JSON
* \*\*Response\*\*: JSON object with post details

#### Get Post Comments API

* \*\*Method\*\*: GET
* \*\*URL\*\*: `/api/posts/<int:post\_id>/comments`
* \*\*Description\*\*: Get all comments for a post
* \*\*Response\*\*: JSON array of comments

[

{

"id": 1,

"content": "Comment content...",

"created\_at": "2025-05-23T18:00:00",

"author": "username",

"user\_id": 1,

"post\_id": 1

}

]

### Error Handling

#### HTTP Status Codes

* \*\*200\*\*: Success
* \*\*302\*\*: Redirect (successful form submission)
* \*\*400\*\*: Bad Request (validation errors)
* \*\*401\*\*: Unauthorized (login required)
* \*\*403\*\*: Forbidden (insufficient permissions)
* \*\*404\*\*: Not Found (resource doesn't exist)
* \*\*500\*\*: Internal Server Error

#### Error Response Format

{

"success": false,

"message": "Error description"

}

# Authentication Flow

## Authentication Flow

### User Registration Flow

1. User visits /register

2. User fills registration form

├── Username (3-64 chars, unique)

├── Email (valid format, unique)

├── Password (min 6 chars)

└── Confirm Password (must match)

3. Form validation (client & server-side)

4. Check username/email uniqueness

5. Hash password using Werkzeug

6. Save user to database

7. Redirect to login page

8. Flash success message

### User Login Flow

1. User visits /login

2. User enters credentials

├── Email

└── Password

3. Server validates credentials

4. Check password hash

5. If valid:

├── Create user session (Flask-Login)

├── Redirect to dashboard

└── Flash welcome message

6. If invalid:

├── Show error message

└── Return to login form

### User Logout Flow

1. User clicks logout

2. Server destroys session

3. Flask-Login logs out user

4. Redirect to home page

5. Flash logout message

### Session Management

#### Flask-Login Integration

* Uses Flask-Login for session management
* Sessions stored server-side
* User ID stored in session cookie
* Automatic login state checking

#### Protected Routes

@login\_required

def protected\_route():

# Only accessible to authenticated users

pass

#### User Loading

@login\_manager.user\_loader

def load\_user(user\_id):

return User.query.get(int(user\_id))

### Security Features

1. \*\*Password Hashing\*\*: Werkzeug PBKDF2 with salt
2. \*\*CSRF Protection\*\*: Flask-WTF tokens
3. \*\*Session Security\*\*: Secure session cookies
4. \*\*Input Validation\*\*: Server-side form validation
5. \*\*Authorization\*\*: Route-level access control

### Authentication Sequence Diagram

User Browser Server Database

| | | |

|-- Registration ---->| | |

| |-- POST /register ->| |

| | |-- Validate form -->|

| | |-- Hash password -->|

| | |-- Save user ------>|

| |<-- Redirect -------|<-- Success --------|

|<-- Success msg -----| | |

| | | |

|-- Login ----------->| | |

| |-- POST /login ---->| |

| | |-- Check user ----->|

| | |<-- User data ------|

| | |-- Verify password->|

| | |-- Create session ->|

| |<-- Set cookie -----| |

|<-- Dashboard -------|<-- Redirect -------| |

# Setup Guide

## Setup Guide

### Prerequisites

#### Required Software

* \*\*Python 3.11+\*\*: Programming language runtime
* \*\*PostgreSQL\*\*: Database system (or SQLite for development)
* \*\*Git\*\*: Version control system
* \*\*Web Browser\*\*: For accessing the application

#### Development Environment

* \*\*Text Editor/IDE\*\*: VS Code, PyCharm, or similar
* \*\*Terminal/Command Prompt\*\*: For running commands

### Installation Steps

#### 1. Clone Repository

git clone <repository-url>

cd csu-ccis-blog

#### 2. Python Environment Setup

# Create virtual environment (recommended)

python -m venv venv

# Activate virtual environment

# On Windows:

venv\Scripts\activate

# On macOS/Linux:

source venv/bin/activate

#### 3. Install Dependencies

pip install -r requirements.txt

Required packages:

* flask
* flask-sqlalchemy
* flask-login
* flask-wtf
* wtforms
* email-validator
* werkzeug
* gunicorn
* psycopg2-binary
* sqlalchemy

#### 4. Environment Variables

Create a `.env` file in the project root:

# Database Configuration

DATABASE\_URL=postgresql://username:password@localhost:5432/csu\_ccis\_blog

# Flask Configuration

SESSION\_SECRET=your-secret-key-here

FLASK\_ENV=development

FLASK\_DEBUG=True

# PostgreSQL Configuration (if using local database)

PGHOST=localhost

PGPORT=5432

PGUSER=your\_username

PGPASSWORD=your\_password

PGDATABASE=csu\_ccis\_blog

#### 5. Database Setup

#### Option A: PostgreSQL (Recommended for Production)

# Install PostgreSQL

# Create database

createdb csu\_ccis\_blog

# The application will automatically create tables on first run

#### Option B: SQLite (Development Only)

# No additional setup required

# Database file will be created automatically

#### 6. Initialize Database

# Run the application to create tables

python main.py

### Running the Application

#### Development Mode

python main.py

The application will be available at: `http://localhost:5000`

#### Production Mode (Gunicorn)

gunicorn --bind 0.0.0.0:5000 --reuse-port --reload main:app

### Environment Configuration

#### Development Settings

* Debug mode enabled
* SQLite database (optional)
* Detailed error messages
* Auto-reload on code changes

#### Production Settings

* Debug mode disabled
* PostgreSQL database
* Error logging
* Gunicorn WSGI server

### Troubleshooting

#### Common Issues

1. \*\*Database Connection Error\*\*

* Verify PostgreSQL is running
* Check DATABASE\_URL configuration
* Ensure database exists

1. \*\*Module Import Errors\*\*

* Activate virtual environment
* Install all dependencies: `pip install -r requirements.txt`

1. \*\*Permission Errors\*\*

* Check file permissions
* Run with appropriate user privileges

1. \*\*Port Already in Use\*\*

* Change port in configuration
* Kill existing process: `lsof -ti:5000 | xargs kill`

#### Debug Steps

1. Check Python version: `python --version`
2. Verify virtual environment is active
3. Check installed packages: `pip list`
4. Review application logs
5. Test database connection

### Project Structure

csu-ccis-blog/

├── app.py # Flask app configuration

├── main.py # Application entry point

├── models.py # Database models

├── routes.py # URL routes

├── forms.py # Form definitions

├── utils.py # Utility functions

├── requirements.txt # Python dependencies

├── .env # Environment variables

├── templates/ # HTML templates

├── static/ # CSS, JS, images

└── docs/ # Documentation

# Usage Guide

## Usage Guide

### Getting Started

#### 1. Account Registration

1. Navigate to the homepage
2. Click "Register" in the navigation menu
3. Fill out the registration form:

* \*\*Username\*\*: Choose a unique username (3-64 characters)
* \*\*Email\*\*: Enter a valid email address
* \*\*Password\*\*: Create a secure password (minimum 6 characters)
* \*\*Confirm Password\*\*: Re-enter the same password

1. Click "Sign Up" to create your account
2. You'll be redirected to the login page with a success message

#### 2. Logging In

1. Click "Login" in the navigation menu
2. Enter your email and password
3. Click "Login" to access your account
4. You'll be redirected to your dashboard

### Core Features

#### Creating Blog Posts

1. \*\*Access the Create Post Form\*\*:

* From the navigation menu, click "New Post"
* Or from your dashboard, click the "New Post" button

1. \*\*Fill Out the Post Form\*\*:

* \*\*Title\*\*: Enter a descriptive title (3-128 characters)
* \*\*Content\*\*: Write your blog post content
* The content supports line breaks and basic formatting

1. \*\*Submit Your Post\*\*:

* Click "Submit" to publish your post
* You'll be redirected to your dashboard with a success message

#### Managing Your Posts

#### Viewing Your Posts

* Access your dashboard from the navigation menu
* See a table of all your posts with:
* Post title (clickable to view full post)
* Creation date
* Last update date
* Number of comments
* Action buttons (Edit/Delete)

#### Editing Posts

1. From your dashboard, click the "Edit" button next to a post
2. Modify the title and/or content
3. Click "Submit" to save changes
4. You'll be redirected to the post detail page

#### Deleting Posts

1. Click the "Delete" button next to a post (dashboard or post detail page)
2. Confirm deletion in the popup modal
3. The post will be permanently removed along with all its comments

#### Commenting System

#### Adding Comments

1. Navigate to any blog post detail page
2. Scroll down to the comments section
3. Enter your comment in the text area
4. Click "Submit" to post your comment
5. Your comment will appear immediately

#### Deleting Comments

* \*\*Your Own Comments\*\*: Click the delete (trash) icon next to any of your comments
* \*\*Comments on Your Posts\*\*: As a post author, you can delete any comment on your posts
* Confirm deletion in the popup modal

#### Navigation and Browsing

#### Homepage

* Displays the latest blog posts from all users
* Shows post previews (first 300 characters)
* Includes pagination for browsing multiple pages
* Shows comment count for each post

#### Post Detail Pages

* Click "Read More" or a post title to view the full post
* See complete post content
* View all comments and add your own
* Access edit/delete options (if you're the author)

#### User Dashboard

* Personal management interface for your content
* Quick access to create new posts
* Overview of all your posts with management options

### User Interface Features

#### Responsive Design

* Mobile-friendly layout that works on all devices
* Touch-friendly buttons and navigation
* Optimized for both desktop and mobile viewing

#### CSU CCIS Branding

* College colors and logo throughout the interface
* Professional appearance suitable for academic use
* Consistent styling with institution guidelines

#### Interactive Elements

* Hover effects on cards and buttons
* Smooth transitions and animations
* Modal dialogs for confirmations
* Real-time form validation

### Data Management

#### CRUD Operations for Posts

* \*\*Create\*\*: New post form with validation
* \*\*Read\*\*: Homepage listing and detail pages
* \*\*Update\*\*: Edit existing posts (authors only)
* \*\*Delete\*\*: Remove posts with confirmation (authors only)

#### CRUD Operations for Comments

* \*\*Create\*\*: Add comments to any post (logged-in users)
* \*\*Read\*\*: View comments on post detail pages
* \*\*Delete\*\*: Remove comments (comment author or post author)

#### User Account Management

* \*\*Create\*\*: Registration with validation
* \*\*Read\*\*: View profile information in navigation
* \*\*Authentication\*\*: Secure login/logout system

### Tips for Best Practices

#### Writing Effective Posts

* Use descriptive, engaging titles
* Structure content with clear paragraphs
* Keep posts focused on specific topics
* Proofread before publishing

#### Community Engagement

* Read and comment on other users' posts
* Provide constructive feedback in comments
* Respect community guidelines and academic standards

#### Account Security

* Use a strong, unique password
* Log out when using shared computers
* Keep your email address current for account recovery

### Troubleshooting

#### Common Issues

* \*\*Can't log in\*\*: Check email and password, ensure account exists
* \*\*Posts not saving\*\*: Ensure all required fields are filled
* \*\*Comments not appearing\*\*: Refresh the page, check internet connection
* \*\*Permission errors\*\*: Ensure you're logged in and accessing your own content

#### Getting Help

* Check that you're logged in for protected features
* Refresh the page if content doesn't load
* Contact system administrator for technical issues

# Code Structure

File and Directory Organization:

|  |  |
| --- | --- |
| **File/Directory** | **Purpose** |
| app.py | Flask application configuration and database setup |
| main.py | Application entry point |
| models.py | SQLAlchemy database models (User, Post, Comment) |
| routes.py | URL routing and view functions |
| forms.py | WTForms form definitions and validation |
| utils.py | Utility functions and decorators |
| templates/ | Jinja2 HTML templates |
| static/css/ | Custom CSS styling with CSU CCIS theme |
| static/js/ | JavaScript for client-side functionality |
| static/images/ | CSU CCIS logo and other images |
| docs/ | Project documentation files |

# Technologies Used

|  |  |  |  |
| --- | --- | --- | --- |
| **Category** | **Technology** | **Version** | **Purpose** |
| Backend | Flask | Latest | Web framework |
| Database | PostgreSQL | Latest | Primary database |
| ORM | SQLAlchemy | Latest | Database abstraction |
| Authentication | Flask-Login | Latest | Session management |
| Forms | Flask-WTF | Latest | Form handling and validation |
| Frontend | Bootstrap 5 | 5.3.0 | CSS framework |
| JavaScript | Vanilla JS | ES6+ | Client-side functionality |
| Server | Gunicorn | Latest | WSGI HTTP server |
| Security | Werkzeug | Latest | Password hashing |

# Key Features Implemented

* User Registration and Authentication with secure password hashing
* Blog Post Management (Create, Read, Update, Delete)
* Comment System for user interaction
* Responsive Design with CSU CCIS branding
* User Dashboard for content management
* Form Validation and Error Handling
* Session Management and Security
* Database Relationships and Data Integrity
* RESTful API Endpoints
* Mobile-Friendly Interface

# Assignment Requirements Met

This project fulfills all requirements for the 21-day course assignment:

* ✓ Simple web application using Flask framework
* ✓ User authentication system (sign-up, sign-in, sign-out)
* ✓ Two CRUD features: Blog Posts and Comments
* ✓ Frontend-Backend-Database integration
* ✓ CSU CCIS branding and professional design
* ✓ Comprehensive documentation package
* ✓ Database design with proper relationships
* ✓ Security best practices implementation
* ✓ Responsive and accessible user interface
* ✓ Production-ready deployment configuration