Visvesvaraya Technological University

Jnana Sangama, Belagavi - 590018

Mini Project Report on (18CSL58)

on

VENTA BLOG

Mini Project Report submitted in partial fulfilment of the requirement for

the award of the degree of

BACHELOR OF ENGINEERING

IN

COMPUTER SCIENCE AND ENGINEERING

Submitted by

P SAI RAM   **1KS18CS065**

Under the guidance of

Dr.DAYANANDA R B   **Mr. KUMAR.K**
Professor   **Assistant Professor**
Department of Computer Science & Engineering
K.S.I.T, Bengaluru-560109

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

K. S. Institute of Technology

#14, Raghuvanahalli, Kanakapura Road, Bengaluru - 560109
2020 - 2021

K. S. Institute of Technology

#14, Raghuvanahalli, Kanakapura Road, Bengaluru - 560109

Department of Computer Science & Engineering

CERTIFICATE

Certified that the Mini Project Work (18CSL58) entitled **VENTA BLOG** is a bonafide work carried out by:

P SAI RAM   **1KS18CS065**

in partial fulfilment for V semester B.E., Mini Project Work in the branch of Computer Science and Engineering prescribed by **Visvesvaraya Technological University, Belagavi** during the period of September 2020 to January 2021. It is certified that all the corrections and suggestions indicated for internal assessment have been incorporated. The Mini Project Work Report has been approved as it satisfies the academic requirements in report of project work prescribed for the Bachelor of Engineering degree.

……………………….   ..……………..………
Signature of the Guide   **Signature of the Guide**

[Dr.Dayananda R B]   **[Kumar.K]**

..………………..………   …...…..……………………
Signature of the HOD   **Signature of the Principal & CEO**
[Dr. Rekha B. Venkatapur]   [Dr. K.V.A. Balaji]

ACKNOWLEDGEMENT

ABSTRACT

The term blogging and blog is a latest buzzword in modern society as more people started reading and writing blogs online. There is a constant increase in the number of people turned in the blogs way and it is a good medium for everybody to write and publish their opinions online. Venta Blog is a blogging application which helps individuals and organizations to create and manage articles in a user friendly way. It enables everyone to create, manage and maintain their individual blogs with a pool of people working on their own blogs. We have created this project to help students to manage their day to day notes taking and writing and updating their schedule accordingly.

TABLE OF CONTENTS

Visvesvaraya Technological University
Jnana Sangama, Belagavi - 590018

A Mini Project Work (18CPL58)
Report on

VENTA BLOG

Mini Project Report submitted in partial fulfilment of the requirement for

the award of the degree of

BACHELOR OF ENGINEERING

IN

COMPUTER SCIENCE AND ENGINEERING

Submitted by

P SAI RAM   **1KS18CS065**

Under the guidance of

Dr. DAYANANDA R B   **Mr. KUMAR.K**
Professor   **Assistant professor**
Department of Computer Science & Engineering
K.S.I.T, Bengaluru-560109

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING
K. S. Institute of Technology
#14, Raghuvanahalli, Kanakapura Road, Bengaluru - 560109
2020 - 2021

CONTENTS

Chapter 1

INTRODUCTION

1.1 OVERVIEW

Venta Blog is a blogging application which helps individuals and organizations to create and manage articles in a user friendly way. It enables everyone to create, manage and maintain their individual blogs with a pool of people working on their own blogs. We have created this project to help students to manage their day to day notes taking and writing and updating their schedule accordingly.

1.2 PROBLEM STATEMENT

The main aim of "Venta blog" is to provide an easy interface for clients to have stable, clean and manageable articles and documents. Where documents are interlinked with each other for effort less information transfer.

1.3 DATABASE MANAGEMENT SYSTEM

A database management system (DBMS) is system software for creating and managing databases. The DBMS provides users and programmers with a systematic way to create, retrieve, update and manage data. The DBMS essentially serves as an interface between the database and end users application programs, ensuring that data is consistently organized and remains easily accessible.

The DBMS manages three important things: the data, the database engine that allows data to be accessed, locked and modified ,and the database schema, which defines the

database's logical structure. These three foundational elements help to provide concurrency, security, data integrity and uniform administration procedures. Typical database administration tasks supported by the DBMS include change management, performance monitoring/tuning and backup and recovery. Many database management systems are also responsible for automated rollbacks, restarts and recovery as well as the logging and auditing of activity.

1.4 SQL

SQL is a standard language for storing, manipulating and retrieving data in databases.

Originally based upon relational algebra and tuple relational calculus, SQL consists of a data definition language, data manipulation language, and data control language. The scope of SQL includes data insert, query, update and delete, schema creation and modification, and data access control.

SQL became a standard of the American National Standards Institute (ANSI) in 1986, and of the International Organization for Standardization (ISO) in 1987.[13]Since then, the standard has been revised to include a larger set of features. Despite the existence of such standards, most SQL code is not completely portable among different database systems without adjustments.

1.5 HTML / JavaScript

HTML is a markup language used for structuring and presenting content on the web and the fifth and current major version of the HTML standard.
HTML5 includes detailed processing models to encourage more interoperable implementations; it extends, improves and rationalizes the markup available for documents, and introduces markup and application programming interfaces (APIs) for complex web applications.

JavaScript often abbreviated as JS, is a high-level, interpreted programming language. It is a language which is also characterized as dynamic, weakly typed, prototype-based and multi-paradigm.

Alongside HTML and CSS, JavaScript is one of the three core technologies of the World Wide Web. JavaScript enables interactive web pages and thus is an essential part of web applications. The vast majority of websites use it, and all major web browsers have a dedicated JavaScript engine to execute it.

1.6 Flask

To create and manage the back-end server we have employed flask as our main framework web framework. Flask is based on the language python as is the majority of our project. Here is a brief description of flask.

Flask is a micro web framework written in Python. It is classified as a microframework because it does not require particular tools or libraries. It has no database abstraction layer, form validation, or any other components where pre-existing third-party libraries provide common functions.

Flask works on the model of model view template (MVT) which is explained in the later part of the report. Flask is based on werkzeug which is a CGI Library.

Flask is a backend framework, so it uses the CGI library to serve some HTTP requests, it adds that header into the response.

Chapter 2

REQUIREMENTS SPECIFICATION

A computerized way of handling information about property and users details is efficient, organized and time saving, compared to a manual way of doing so. This is done through a database driven web application whose requirements are mentioned in this section.

2.1 OVERALL DESCRIPTION

A reliable and scalable database driven web application with security features that is easy to use and maintain is the requisite.

2.2 SPECIFIC REQUIREMENTS

The specific requirements of the Venta Blog are stated as follows:

2.2.1 SOFTWARE REQUIREMENTS

- ☐ TextEditor/IDE - Visual Studio Code
- ☐ Web Browser – Firefox 50 or later, Google Chrome – 60 or later
- ☐ Database support - MySQL 5.7
  - o MySQL Server 5.7
- ☐ Operating system – Windows 7 or later / Ubuntu 16.04 or later
- ☐ Python2.8 or later (python3.8 recommended)
- ☐ Server deployment - Python WSGI HTTP server / Nginx server / SQL server

2.2.2 HARDWARE REQUIREMENTS

- ☐ Processor – Pentium IV or above
- ☐ RAM – 2 GB or more
- ☐ Hard disk – 3 GB or more
- ☐ Monitor – VGA of 1024x768 screen resolution
- ☐ Keyboard and Mouse

2.2.3 TECHNOLOGY

☐ HTML is used for the front end design. It provides a means to structure text based information in a document. It allows users to produce web pages that include text, graphics and hyperlinks.

☐ CSS (Cascading Style Sheets) is a style sheet language used for describing the presentation of a document written in a markup language. Although most often used to set the visual style of web pages and user interfaces written in HTML and XHTML, the language can be applied to any XML document.

☐ SQL is the language used to manipulate relational databases. It is tied closely with the relational model. It is issued for the purpose of data definition and data manipulation.

☐ Flask backend framework is a simple yet powerful technology for creating and maintaining dynamic-content web pages. It is based on the Python programming language. It can be thought of as an extension to servlet because it provides more functionality than servlet A jinja template page consists of HTML tags and jinja tags. The template pages are easier to maintain than servlet because we can separate designing and development.

☐ We require a connection between the front end and back end components to write to the database and fetch required data.

Chapter 3

3. DETAILED DESIGN

3.1 SYSTEM DESIGN

The web server needs a Jinja **template engine**, i.e., a container to process jinja template pages. The flask backend server is responsible for sending requests to the jinja template engine. A jinja template works with the flask web server to provide the runtime environment and other services a template needs to be rendered. It knows how to understand the special elements that are part of the template. This server will act as a mediator between the client browser and a database. This working is similar to how ajax requests are sent and received.

The following diagram shows a very basic view of working on the backend server.
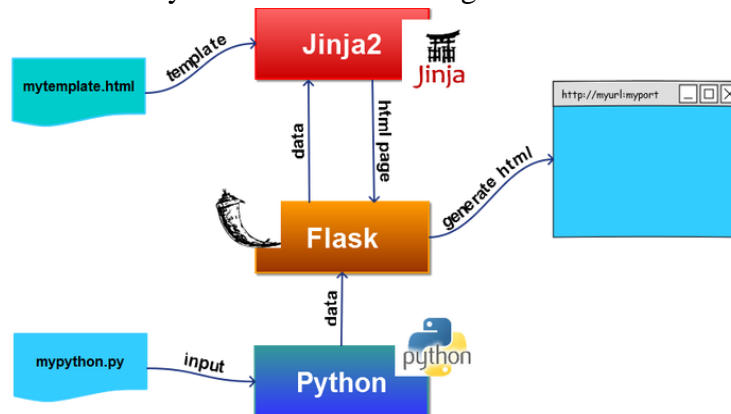

Fig. 3.1 Working of the system

Three-tier Client / Server database architecture is commonly used architecture for web applications. Intermediate layer called Application server or Web Server stores the web connectivity software and the business logic (constraints) part of application used to access the right amount of data from the database server. This layer acts like a medium for sending partially processed data between the database server and the client. Database architecture focuses on the design, development, implementation and maintenance of computer programs that store and organize information for businesses, agencies and institutions. A database architect develops and implements software to meet the needs of users. Several types of databases, including relational or multimedia, may be created. Additionally, database architects may use one of several languages to create databases, such as structured query language.

3.2 ENTITY RELATIONSHIP DIAGRAM

An entity–relationship model is usually the result of systematic analysis to define and describe what is important to processes in an area of a business.

An E-R model does not define the business processes; it only presents a business data schema in graphical form. It is usually drawn in a graphical form as boxes (entities) that are connected by lines (relationships) which express the associations and dependencies between entities.

Entities may be characterized not only by relationships, but also by additional properties (attributes), which include identifiers called "primary keys". Diagrams created to represent attributes as well as entities and relationships may be called entity-attribute-relationship diagrams, rather than entity-relationship models.

An ER model is typically implemented as a database. In a simple relational database implementation, each row of a table represents one instance of an entity type, and each field in a table represents an attribute type. In a relational database a relationship between entities is implemented by storing the primary key of one entity as a pointer or "foreign key" in the table of another entity.

There is a tradition for ER/data models to be built at two or three levels of abstraction. Note that the conceptual-logical-physical hierarchy below is used in other kinds of specification, and is different from the three schema approach to software engineering. While useful for organizing data that can be represented by a relational structure, an entity-relationship diagram can't sufficiently represent semi-structured or unstructured data, and an ER Diagram is unlikely to be helpful on its own in integrating data into a pre-existing information system.

Cardinality notations define the attributes of the relationship between the entities. Cardinalities can denote that an entity is optional.
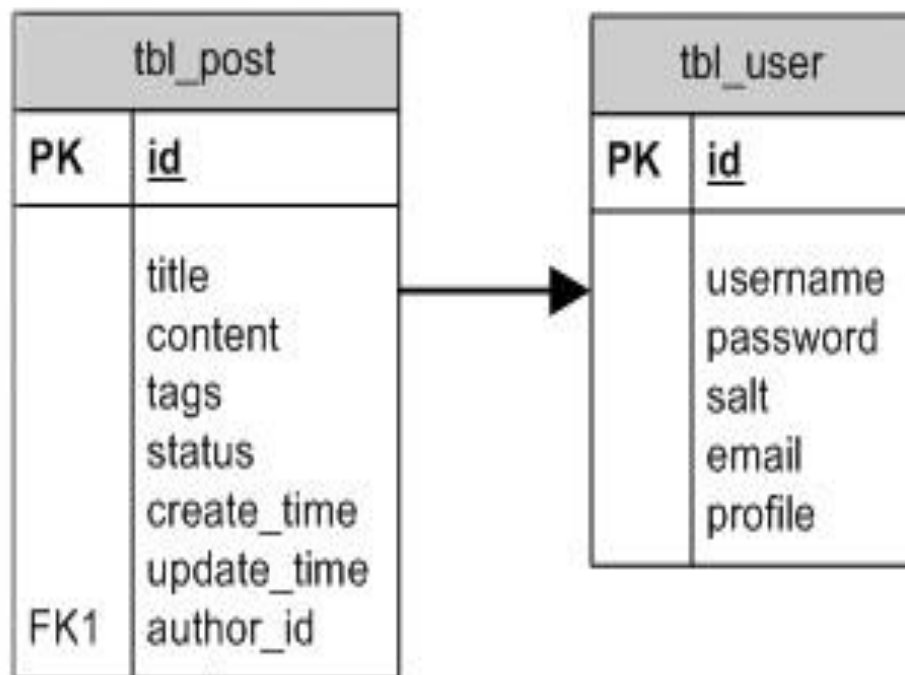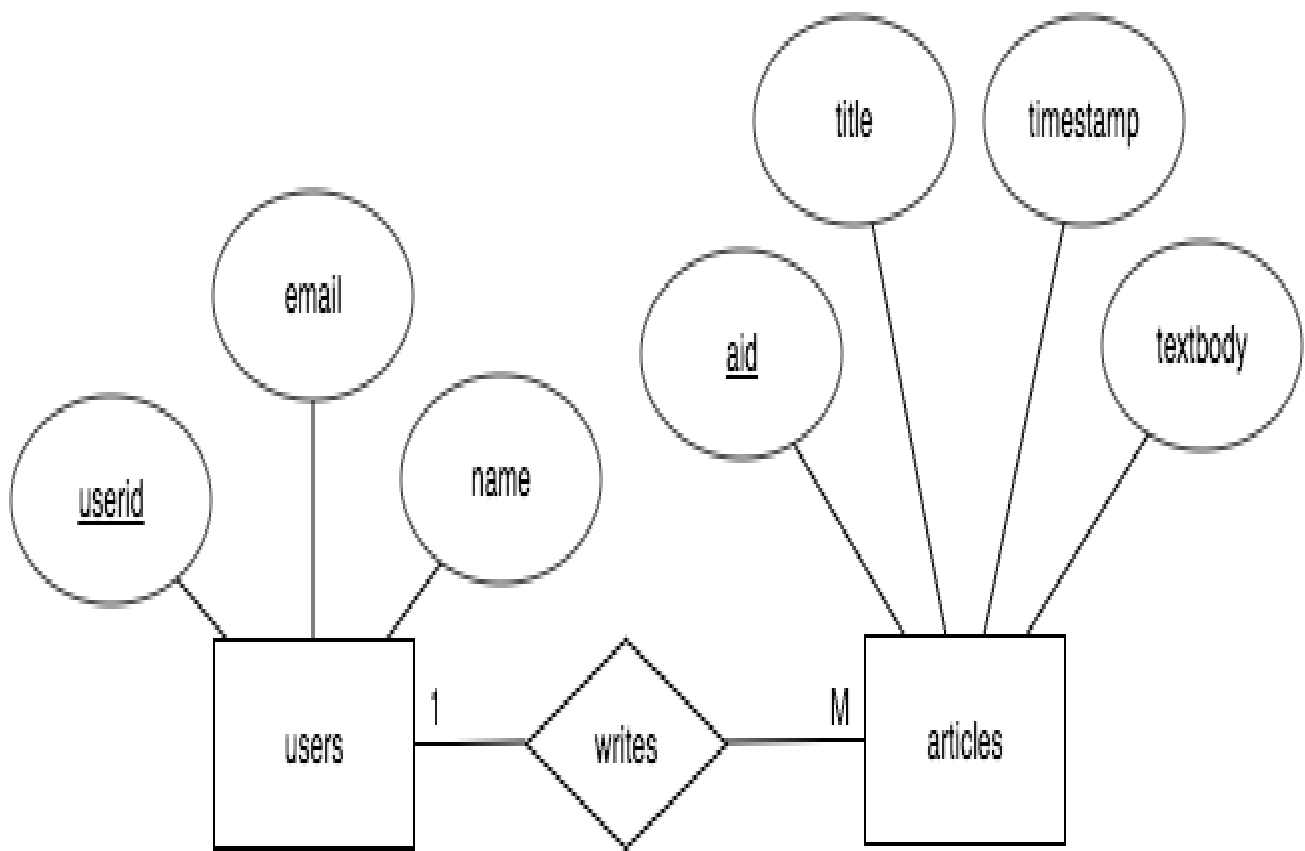
Fig. 3.2.1 Enhanced ER diagram of Vent Blog

Fig. 3.2.2, ER diagram of Venta Blog

3.3 RELATIONAL SCHEMA

The term "schema" refers to the organization of data as a blueprint of how the database is constructed. The formal definition of a database schema is a set of formulas called integrity constraints imposed on a database. A relational schema shows references among fields in the database. When a primary key is referenced in another table in the database, it is called a foreign key. This is denoted by an arrow with the head pointing at the referenced key attribute. A schema diagram helps organize values in the database. The following diagram shows the schema diagram for the database.

Users:

User_id	User Name	email	password	image file	Posts

Posts:

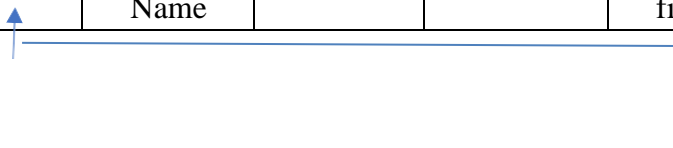| Post_id	Post_title	date_posted	content	User_id

Fig. 3.3, Schema diagram

3.4 DESCRIPTION OF TABLES

The database consists of six tables:

1. Users: It stores the user details.
   - ☐ User_id: Unique user id done by auto increment.
   - ☐ User Name: Name of the user.
   - ☐ Email: Email id of the user.
   - ☐ Profile Image: Profile Image of the user.
   - ☐ Password: Password associated with user to login into system
   - ☐ Posts: Posts associated with user

2. Posts: It stores the posts created by the users.
   - ☐ Post_id: The id associated with the post.
   - ☐ Title: The title of the post.
   - ☐ Date_posted: The date on which the post was posted.
   - ☐ Content: The content of the Post
   - ☐ User_id: The id of the user that created this post.

Dept. of CSE, K.S. Institute of Technology

Chapter 4

IMPLEMENTATION

4.1 MODULES AND THEIR ROLES

4.1.1 routes.py: The api for mapping urls to flask api

```
@app.route("/")
@app.route("/home")
def home():
    page = request.args.get('page', 1, type=int)
    posts = Post.query.order_by(Post.date_posted.desc()).paginate(page=page, per_page=5)
    return render_template('home.html', posts=posts)



@app.route("/about")
def about():
    return render_template('about.html', title='About')



@app.route("/register", methods=['GET', 'POST'])
def register():
    if current_user.is_authenticated:
        return redirect(url_for('home'))
    form = RegistrationForm()
    if form.validate_on_submit():
        hashed_password = bcrypt.generate_password_hash(form.password.data).decode('utf-8')
        user = User(username=form.username.data, email=form.email.data,
password=hashed_password)
        db.session.add(user)
        db.session.commit()
        flash('Your account has been created! You are now able to log in', 'success')
        return redirect(url_for('login'))
    return render_template('register.html', title='Register', form=form)
```

```python
@app.route("/login", methods=['GET', 'POST'])
def login():
    if current_user.is_authenticated:
        return redirect(url_for('home'))
    form = LoginForm()
    if form.validate_on_submit():
        user = User.query.filter_by(email=form.email.data).first()
        if user and bcrypt.check_password_hash(user.password, form.password.data):
            login_user(user, remember=form.remember.data)
            next_page = request.args.get('next')
            return redirect(next_page) if next_page else redirect(url_for('home'))
        else:
            flash('Login Unsuccessful. Please check email and password', 'danger')
    return render_template('login.html', title='Login', form=form)


@app.route("/logout")
def logout():
    logout_user()
    return redirect(url_for('home'))


def save_picture(form_picture):
    random_hex = secrets.token_hex(8)
    _, f_ext = os.path.splitext(form_picture.filename)
    picture_fn = random_hex + f_ext
    picture_path = os.path.join(app.root_path, 'static/profile_pics', picture_fn)

    output_size = (125, 125)
    i = Image.open(form_picture)
    i.thumbnail(output_size)
    i.save(picture_path)
    return picture_fn


@app.route("/account", methods=['GET', 'POST'])
@login_required
def account():
    form = UpdateAccountForm()
    if form.validate_on_submit():
        if form.picture.data:
            picture_file = save_picture(form.picture.data)
            current_user.image_file = picture_file
        current_user.username = form.username.data
        current_user.email = form.email.data
```

```python
        db.session.commit()
        flash('Your account has been updated!', 'success')
        return redirect(url_for('account'))
    elif request.method == 'GET':
        form.username.data = current_user.username
        form.email.data = current_user.email
    image_file = url_for('static', filename='profile_pics/' + current_user.image_file)
    return render_template('account.html', title='Account',
                image_file=image_file, form=form)


@app.route("/post/new", methods=['GET', 'POST'])
@login_required
def new_post():
    form = PostForm()
    if form.validate_on_submit():
        post = Post(title=form.title.data, content=form.content.data, author=current_user)
        db.session.add(post)
        db.session.commit()
        flash('Your post has been created!', 'success')
        return redirect(url_for('home'))
    return render_template('create_post.html', title='New Post',
                form=form, legend='New Post')


@app.route("/post/<int:post_id>")
def post(post_id):
    post = Post.query.get_or_404(post_id)
    return render_template('post.html', title=post.title, post=post)


@app.route("/post/<int:post_id>/update", methods=['GET', 'POST'])
@login_required
def update_post(post_id):
    post = Post.query.get_or_404(post_id)
    if post.author != current_user:
        abort(403)
    form = PostForm()
    if form.validate_on_submit():
        post.title = form.title.data
        post.content = form.content.data
        db.session.commit()
        flash('Your post has been updated!', 'success')
        return redirect(url_for('post', post_id=post.id))
```

```python
    elif request.method == 'GET':
        form.title.data = post.title
        form.content.data = post.content
    return render_template('create_post.html', title='Update Post',
                    form=form, legend='Update Post')


@app.route("/post/<int:post_id>/delete", methods=['POST'])
@login_required
def delete_post(post_id):
    post = Post.query.get_or_404(post_id)
    if post.author != current_user:
        abort(403)
    db.session.delete(post)
    db.session.commit()
    flash('Your post has been deleted!', 'success')
    return redirect(url_for('home'))


@app.route("/user/<string:username>")
def user_posts(username):
    page = request.args.get('page', 1, type=int)
    user = User.query.filter_by(username=username).first_or_404()
    posts = Post.query.filter_by(author=user)\
        .order_by(Post.date_posted.desc())\
        .paginate(page=page, per_page=5)
    return render_template('user_posts.html', posts=posts, user=user)



def send_reset_email(user):
    token = user.get_reset_token()
    msg = Message('Password Reset Request',
            sender='noreply@demo.com',
            recipients=[user.email])
    msg.body = f'''To reset your password, visit the following link:
{url_for('reset_token', token=token, _external=True)}

If you did not make this request then simply ignore this email and no changes will be made.
'''
    mail.send(msg)
```

```python
@app.route("/reset_password", methods=['GET', 'POST'])
def reset_request():
    if current_user.is_authenticated:
        return redirect(url_for('home'))
    form = RequestResetForm()
    if form.validate_on_submit():
        user = User.query.filter_by(email=form.email.data).first()
        send_reset_email(user)
        flash('An email has been sent with instructions to reset your password.', 'info')
        return redirect(url_for('login'))
    return render_template('reset_request.html', title='Reset Password', form=form)


@app.route("/reset_password/<token>", methods=['GET', 'POST'])
def reset_token(token):
    if current_user.is_authenticated:
        return redirect(url_for('home'))
    user = User.verify_reset_token(token)
    if user is None:
        flash('That is an invalid or expired token', 'warning')
        return redirect(url_for('reset_request'))
    form = ResetPasswordForm()
    if form.validate_on_submit():
        hashed_password = bcrypt.generate_password_hash(form.password.data).decode('utf-8')
        user.password = hashed_password
        db.session.commit()
        flash('Your password has been updated! You are now able to log in', 'success')
        return redirect(url_for('login'))
    return render_template('reset_token.html', title='Reset Password', form=form)
```

4.1.2 Models.py: All the ORM classes.

```python
@login_manager.user_loader
def load_user(user_id):
    return User.query.get(int(user_id))


class User(db.Model, UserMixin):
    id = db.Column(db.Integer, primary_key=True)
    username = db.Column(db.String(20), unique=True, nullable=False)
    email = db.Column(db.String(120), unique=True, nullable=False)
    image_file = db.Column(db.String(20), nullable=False, default='default.jpg')
    password = db.Column(db.String(60), nullable=False)
    posts = db.relationship('Post', backref='author', lazy=True)



    def get_reset_token(self, expires_sec=1800):
        s = Serializer(app.config['SECRET_KEY'], expires_sec)
        return s.dumps({'user_id': self.id}).decode('utf-8')


    @staticmethod
    def verify_reset_token(token):
        s = Serializer(app.config['SECRET_KEY'])
        try:
            user_id = s.loads(token)['user_id']
        except:
            return None
        return User.query.get(user_id)


    def __repr__(self):
        return f"User('{self.username}', '{self.email}', '{self.image_file}')"
```

```python
class Post(db.Model):
    id = db.Column(db.Integer, primary_key=True)
    title = db.Column(db.String(100), nullable=False)
    date_posted = db.Column(db.DateTime, nullable=False, default=datetime.utcnow)
    content = db.Column(db.Text, nullable=False)
    user_id = db.Column(db.Integer, db.ForeignKey('user.id'), nullable=False)


    def __repr__(self):
        return f"Post('{self.title}', '{self.date_posted}')"
```

4.1.3 Forms.py - API for getting data from the html form.

```python
from flaskblog.models import User
from flask_wtf import FlaskForm
from flask_wtf.file import FileField, FileAllowed
from flask_login import current_user
from wtforms import StringField, PasswordField, SubmitField, BooleanField, TextAreaField
from wtforms.validators import DataRequired, Length, Email, EqualTo, ValidationError


class RegistrationForm(FlaskForm):
    username = StringField('Username',
                validators=[DataRequired(), Length(min=2, max=20)])
    email = StringField('Email',
                validators=[DataRequired(), Email()])
    password = PasswordField('Password', validators=[DataRequired()])
    confirm_password = PasswordField('Confirm Password',
                validators=[DataRequired(), EqualTo('password')])
    submit = SubmitField('Sign Up')
```

```python
def validate_username(self, username):
    user = User.query.filter_by(username=username.data).first()
    if user:
        raise ValidationError('That username is taken. Please choose a different one.')


def validate_email(self, email):
    user = User.query.filter_by(email=email.data).first()
    if user:
        raise ValidationError('That email is taken. Please choose a different one.')

class LoginForm(FlaskForm):
    email = StringField('Email',
                validators=[DataRequired(), Email()])
    password = PasswordField('Password', validators=[DataRequired()])
    remember = BooleanField('Remember Me')
    submit = SubmitField('Login')



class UpdateAccountForm(FlaskForm):
    username = StringField('Username',
                    validators=[DataRequired(), Length(min=2, max=20)])
    email = StringField('Email',
                validators=[DataRequired(), Email()])
    picture = FileField('Update Profile Picture', validators=[FileAllowed(['jpg', 'png'])])
    submit = SubmitField('Update')


    def validate_username(self, username):
        if username.data != current_user.username:
            user = User.query.filter_by(username=username.data).first()
            if user:
                raise ValidationError('That username is taken. Please choose a different one.')
```

```python
    def validate_email(self, email):
        if email.data != current_user.email:
            user = User.query.filter_by(email=email.data).first()
            if user:
                raise ValidationError('That email is taken. Please choose a different one.')


class PostForm(FlaskForm):
    title = StringField('Title', validators=[DataRequired()])
    content = TextAreaField('Content', validators=[DataRequired()])
    submit = SubmitField('Post')


class RequestResetForm(FlaskForm):
    email = StringField('Email',
                        validators=[DataRequired(), Email()])
    submit = SubmitField('Request Password Reset')


    def validate_email(self, email):
        user = User.query.filter_by(email=email.data).first()
        if user is None:
            raise ValidationError('There is no account with that email. You must register first.')


class ResetPasswordForm(FlaskForm):
    password = PasswordField('Password', validators=[DataRequired()])
    confirm_password = PasswordField('Confirm Password',
                        validators=[DataRequired(), EqualTo('password')])
    submit = SubmitField('Reset Password')
```

4.1.4 __init__.py - The ground configuration of the app.

```python
import os

from flask import Flask

from flask_sqlalchemy import SQLAlchemy

from flask_bcrypt import Bcrypt

from flask_login import LoginManager

from flask_mail import Mail


app = Flask(__name__)

app.config['SECRET_KEY'] = '5791628bb0b13ce0c676dfde280ba245'

app.config['SQLALCHEMY_DATABASE_URI'] = 'sqlite:///site.db'

db = SQLAlchemy(app)

bcrypt = Bcrypt(app)

login_manager = LoginManager(app)

login_manager.login_view = 'login'

login_manager.login_message_category = 'info'

app.config['MAIL_SERVER'] = 'smtp.googlemail.com'

app.config['MAIL_PORT'] = 587

app.config['MAIL_USE_TLS'] = True

app.config['MAIL_USERNAME'] = os.environ.get('EMAIL_USER')

app.config['MAIL_PASSWORD'] = os.environ.get('EMAIL_PASS')

mail = Mail(app)


from flaskblog import routes
```

4.2 RESULT

The resulting system is able to:

- ☐ Authenticate user credentials during login.

- ☐ Salted encryption for security of user passwords.

- ☐ Register new users and link to their banks.

- ☐ Allow users to view posts of all users.

- ☐ Allow users to create, delete and update their own posts.

Chapter 5

TESTING

5.1 SOFTWARE TESTING

Testing is the process used to help identify correctness, completeness, security and quality of developed software. This includes executing a program with the intent of finding errors. It is important to distinguish between faults and failures. Software testing can provide objective, independent information about the quality of software and risk of its failure to users or sponsors. It can be conducted as soon as executable software (even if partially complete) exists. Most testing occurs after system requirements have been defined and then implemented in testable programs.

5.2 MODULE TESTING AND INTEGRATION

Module testing is a process of testing the individual subprograms, subroutines, classes, or procedures in a program. Instead of testing whole software program at once, module testing recommend testing the smaller building blocks of the program. It is largely white box oriented. The objective of doing Module testing is not to demonstrate proper functioning of the module but to demonstrate the presence of an error in the module. Module testing allows implementing of parallelism into the testing process by giving the opportunity to test multiple modules simultaneously.

The final integrated system too has been tested for various test cases such as duplicate entries and type mismatch.

Dept. of CSE, K.S. Institute of Technology

5.3 LIMITATIONS

☐ Does not allow creation of posts with image(SQL limitation).

☐ Better management of links after logging in.

Only few fields available for content entry.

Chapter 6

SNAPSHOTS

This chapter consists of working screenshots of the project.

6.1 LOGIN PAGE

This is the login page for existing users and is the first page shown to any customer.

6.2 REGISTRATION PAGE

This is the registration page for any new user

6.3 HOME PAGE
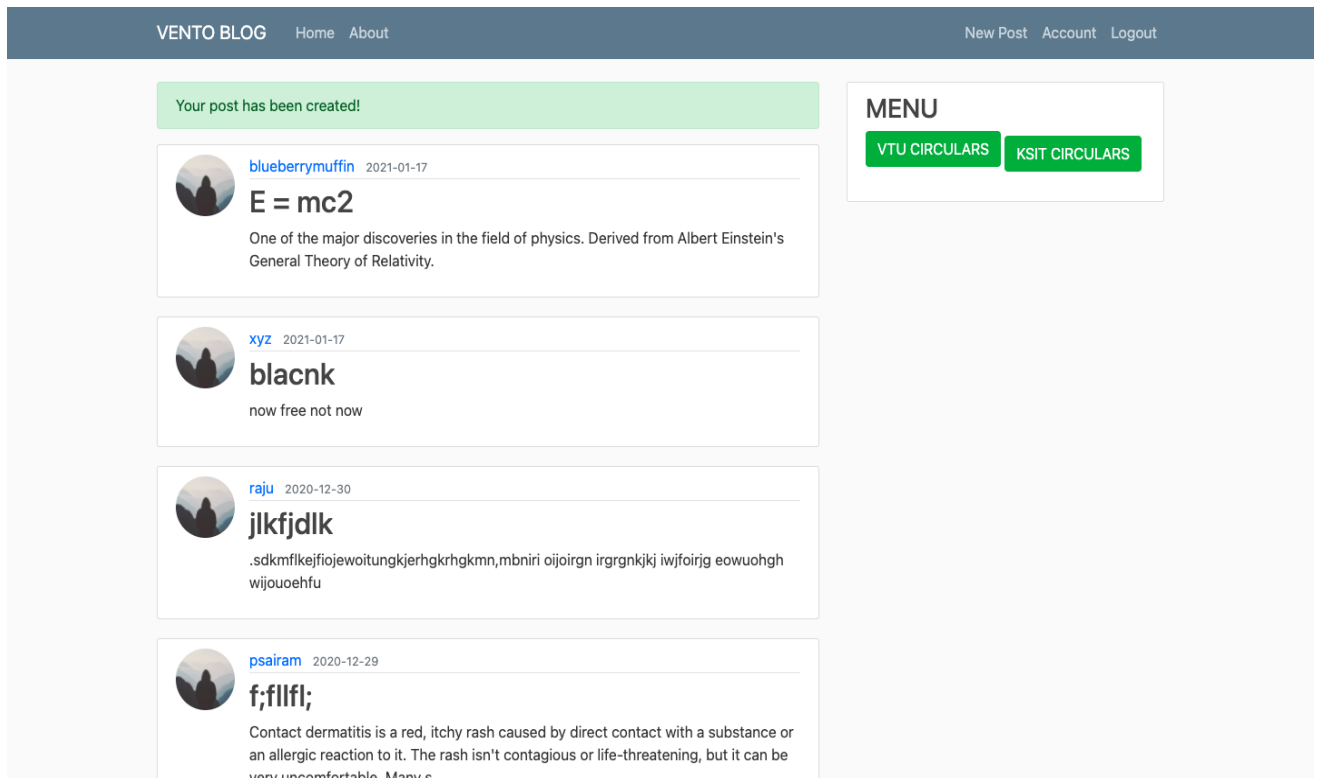
First home page shown to customers after login.

6.4 CREATE POST PAGE
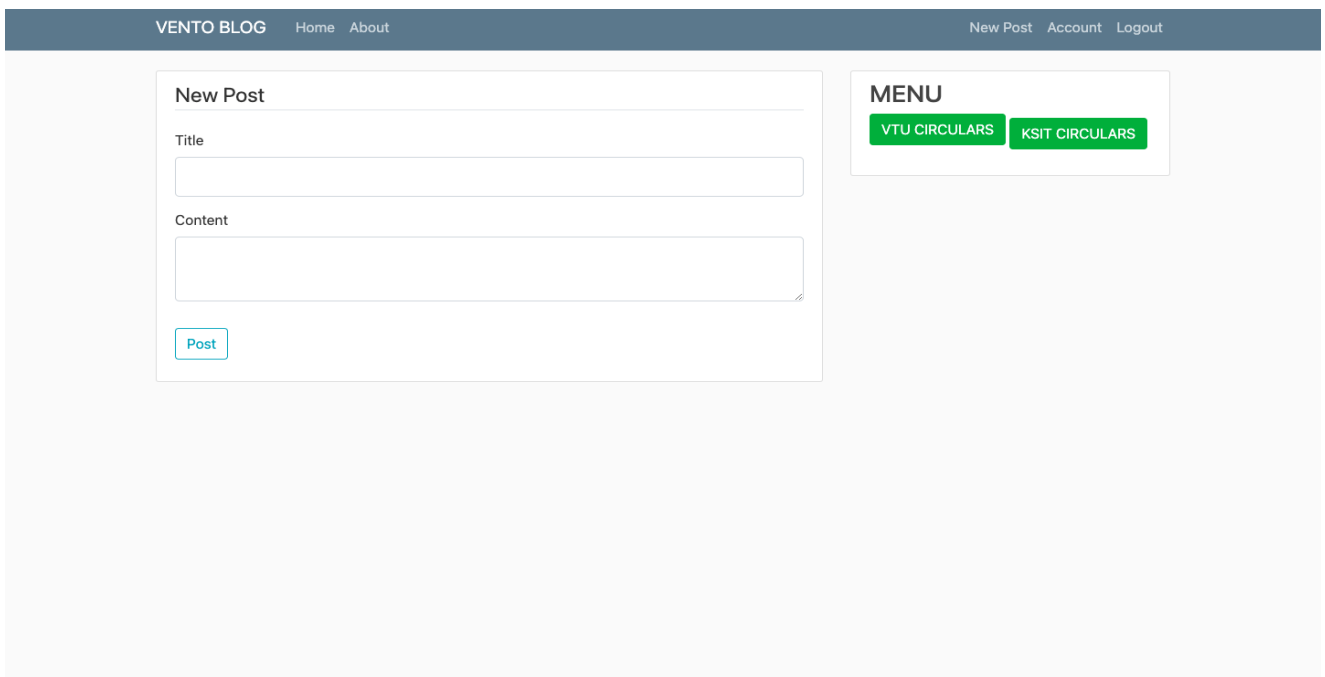
6.5 UPDATE PROFILE PAGE

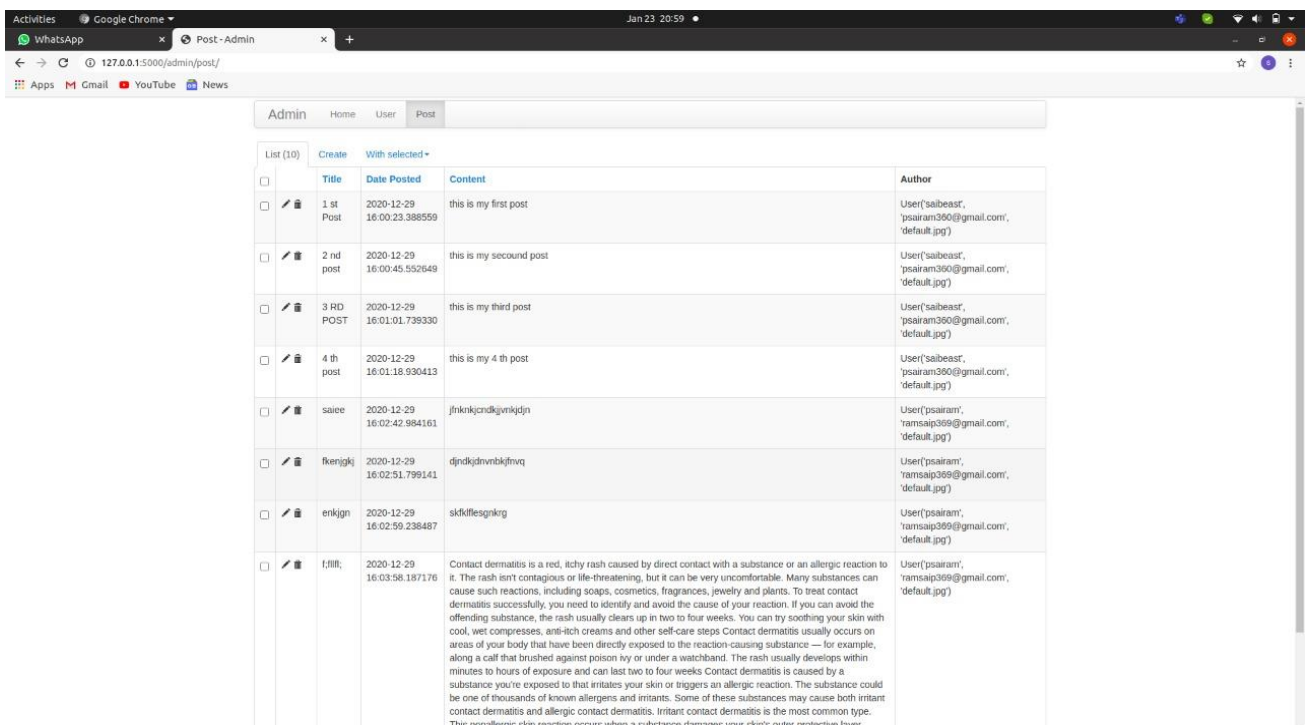This page is for updating the profile details of the user.

6.6 POST UPDATION AND DELETION PAGE

6.7 ADMIN INTERFACE
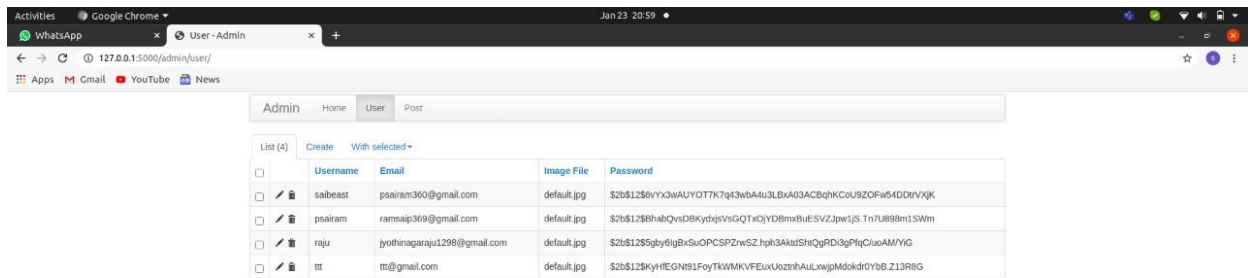
6.8 ADMIN POSTS PAGE

CONCLUSION

Venta Blog is a blogging application which helps individuals and organizations to create and manage articles in a user friendly way. It enables everyone to create, manage and maintain their individual blogs with a pool of people working on their own blogs. We have created this project to help students to manage their day to day notes taking and writing and updating their schedule accordingly. This is developed using HTML5, CSS, JavaScript, Python3, Flask, SQL. The goals achieved by this project are:

☐ 1. Centralized database

☐ 2. Easier creation, updation, sharing and deletion of various articles.

☐ 3. User friendly environment.

☐ 4. Efficient management of blogs.

☐ 5. Ability to view historical data and analyse them for better production.

Dept. of CSE, K.S. Institute of Technology

FUTURE ENHANCEMENTS

Future upgrades to this project will implement:

- ☐ Better interfaces for the ability to view the of various companies including better analytics, more data across various companies, sectors and industries

- ☐ More stock market platforms including Sensex, Dow Jones etc.

- ☐ Ability to trade in forex exchanges and mutual funds.

- ☐ Better banking implementations between the customer and his bank.

- ☐ Ability to see and analyse the various companies customers tend to trade and analyse these for better info.

- ☐ Ability to view timely data across various years and months between various time ranges as required.

REFERENCES

1. Ramakrishnan, R., & Gehrke, J. (2011). Database management systems. Boston: McGraw-Hill.

2. Monson-Haefel, R. (2007). J2EE Web services. Boston, Mass: Addison-Wesley. Silberschatz A., Korth H. F., & Sudarshan S. (2011).

3. Database systems concepts. Estados Unidos: McGraw-Hill Companies, Inc.

4. Hanna P. (2002): JSP 2.0 The Complete Reference, Second Edition McGraw Hill Education.

5. David F. (2011). JavaScript: The Definitive Guide Sixth edition.

6. https://www.w3schools.com

7. https://www.canvasjs.com

8. https://getbootstrap.com/

9. https://fontawesome.com