ASHA YACOB 20234895
MARWA MURSI 20234719
ISMAIL YAQUB 20227028
RANA AHMED 20234869

Project Report: TechCult Blog Application

Overview

The TechCult Blog application is a desktop-based blogging platform designed using Python's PyQt5 framework. This application allows users to log in, create posts, comment on posts, and manage their profile information. The project provides a user-friendly interface with various interactive elements such as buttons, text fields, and dialogs to enhance the user experience.

Key Features

User Authentication:

Login and logout functionality to manage user sessions.
Profile management to update email and bio information.
Post Management:

Creating, viewing, and deleting posts.
Adding comments to posts.
Liking and disliking posts with unique user tracking.
Interactive User Interface:

Responsive buttons and input fields.
Scrollable area for viewing multiple posts.
Dialogs for profile management and post creation.
Detailed Explanation of the Code

Imports and Setup

pythonCopy code

```
import sys
from PyQt5 import QtWidgets, QtGui, QtCore
from PyQt5.QtWidgets import QMessageBox, QTextEdit, QPushButton, QVBoxLayout, QLineEdit, QHBoxLayout, QLabel
from datetime import datetime
import warnings

warnings.filterwarnings("ignore", category=DeprecationWarning)
```

sys: Provides access to some variables used or maintained by the Python interpreter.
PyQt5: A set of Python bindings for the Qt application framework, allowing the creation of GUI applications.
datetime: Used to handle date and time operations.

sys: Provides access to some variables used or maintained by the Python interpreter.
PyQt5: A set of Python bindings for the Qt application framework, allowing the creation of GUI applications.
datetime: Used to handle date and time operations.
warnings: Used to control warning messages.
Main Application Class
pythonCopy code

```
class App(QtWidgets.QWidget):
def __init__(self):
super().__init__()
self.setWindowTitle("TechCult Blog")
self.setGeometry(100, 100, 700, 500)
self.setStyleSheet("""
background-color: qlineargradient(spread:pad, x1:0, y1:0.193, x2:0.829, y2:0.301136, stop:0 rgba(66, 183, 255, 255), stop:1 rgba(232, 43, 255, 255));
border-radius: 25px;
font-size: 16px; /* Set default font size */
""")

self.logged_in = False
self.current_user = None
self.posts = []
self.user_email = ""
self.user_bio = ""

self.create_widgets()
```

App class: Inherits from QtWidgets.QWidget and initializes the main window with a title, size, and style.
Attributes: Manages the application state, including login status, current user, posts, email, and bio.
Creating Widgets
pythonCopy code

```
def create_widgets(self):
self.login_button = QPushButton("Login")
self.login_button.clicked.connect(self.login)
self.login_button.setStyleSheet("""
background-color: black;
color: white;
border-radius: 15px;
padding: 10px 20px;
""")
```

```
self.logout_button = QPushButton("Logout")
self.logout_button.clicked.connect(self.logout)
self.logout_button.setStyleSheet("""
background-color: black;
color: white;
border-radius: 15px;
padding: 10px 20px;
""")
self.logout_button.hide()

self.profile_button = QPushButton("Profile Page")
self.profile_button.clicked.connect(self.show_profile)
self.profile_button.setStyleSheet("""
background-color: black;
color: white;
border-radius: 15px;
padding: 10px 20px;
""")
self.profile_button.hide()

self.posts_layout = QVBoxLayout()

self.username_edit = QLineEdit()
self.username_edit.setPlaceholderText("Enter username")
self.username_edit.setStyleSheet("font-size: 16px;") # Set font size for QLineEdit

layout = QVBoxLayout()
layout.addWidget(self.username_edit)
layout.addWidget(self.login_button)
layout.addWidget(self.logout_button)
layout.addWidget(self.profile_button)

self.posts_scroll = QtWidgets.QScrollArea()
self.posts_scroll.setWidgetResizable(True)
self.posts_widget = QtWidgets.QWidget()
self.posts_widget.setLayout(self.posts_layout)
self.posts_scroll.setWidget(self.posts_widget)

layout.addWidget(self.posts_scroll)
```

```
self.create_post_button = QPushButton("Create Post")
self.create_post_button.clicked.connect(self.create_post)
self.create_post_button.setStyleSheet("""
background-color: black;
color: white;
border-radius: 15px;
padding: 10px 20px;
""")

layout.addWidget(self.create_post_button)

layout.setContentsMargins(25, 25, 25, 25)
self.setLayout(layout)
```

create_widgets() method: Initializes all the widgets used in the application, including buttons, text fields, and layout containers.
Style Sheets: Applied to buttons for consistent styling.
User Authentication Methods
pythonCopy code

```
def login(self):
username = self.username_edit.text().strip()
if not username:
QMessageBox.information(self, "Login", "Please enter username.")
return

if not self.logged_in:
self.logged_in = True
self.current_user = username
QMessageBox.information(self, "Login", f"Welcome, {username}!")
self.update_posts()
self.login_button.hide()
self.logout_button.show()
self.profile_button.show()
else:
QMessageBox.information(self, "Login", "You are already logged in!")

def logout(self):
self.logged_in = False
self.current_user = None
self.login_button.show()
```

```
button.parentWidget().deleteLater()
self.posts.pop(post_index)

def add_comment(self, button):
post_index = self.posts_layout.indexOf(button.parentWidget())
if post_index != -1:
comment, ok = QtWidgets.QInputDialog.getText(self, "Add Comment", "Enter your comment:")
if ok:
self.posts[post_index]["comments"].append({"author": self.current_user, "content": comment})
self.update_posts()

def like_post(self, button):
post_index = self.posts_layout.indexOf(button.parentWidget())
if post_index != -1:
post = self.posts[post_index]
if self.current_user not in post["likes"]:
post["likes"].add(self.current_user)
if self.current_user in post["dislikes"]:
post["dislikes"].remove(self.current_user)
self.update_posts()
else:
QMessageBox.information(self, "Like", "You have already liked this post.")

def dislike_post(self, button):
post_index = self.posts_layout.indexOf(button.parentWidget())
if post_index != -1:
post = self.posts[post_index]
if self.current_user not in post["dislikes"]:
post["dislikes"].add(self.current_user)
if self.current_user in post["likes"]:
post["likes"].remove(self.current_user)
self.update_posts()
else:
QMessageBox.information(self, "Dislike", "You have already disliked this post.")
```

delete_post(): Deletes a specific post.
add_comment(): Adds a comment to a specific post.
like_post(): Likes a specific post and handles like-dislike toggling.
dislike_post(): Dislikes a specific post and handles dislike-like toggling.
Updating the UI

```
def logout(self):
self.logged_in = False
self.current_user = None
self.login_button.show()
self.logout_button.hide()
self.profile_button.hide()
```

login() method: Handles user login, updates the UI and application state.
logout() method: Handles user logout and resets the UI and application state.
Post Management Methods
pythonCopy code

```
def create_post(self):
if not self.logged_in:
QMessageBox.information(self, "Create Post", "You need to log in first!")
return

title, ok1 = QtWidgets.QInputDialog.getText(self, "Create Post", "Enter post title:")
content, ok2 = QtWidgets.QInputDialog.getText(self, "Create Post", "Enter post content:")
topic, ok3 = QtWidgets.QInputDialog.getText(self, "Create Post", "Enter post topic:")

if ok1 and ok2 and ok3:
post = {
"title": title,
"content": content,
"topic": topic, # Add the topic to the post dictionary
"author": self.current_user,
"likes": set(), # Initialize likes as a set to store unique user IDs
"dislikes": set(), # Initialize dislikes as a set to store unique user IDs
"comments": [],
"timestamp": datetime.now().strftime('%Y-%m-%d %H:%M:%S')
}
self.posts.append(post)
self.update_posts()
```

create_post() method: Allows users to create a new post with title, content, and topic, storing it in the posts list.
Post Interaction Methods
pythonCopy code

```
def delete_post(self, button):
post_index = self.posts_layout.indexOf(button.parentWidget())
```

```
def like_post(self, button):
post_index = self.posts_layout.indexOf(button.parentWidget())
if post_index != -1:
post = self.posts[post_index]
if self.current_user not in post["likes"]:
post["likes"].add(self.current_user)
if self.current_user in post["dislikes"]:
post["dislikes"].remove(self.current_user)
self.update_posts()
else:
QMessageBox.information(self, "Like", "You have already liked this post.")

def dislike_post(self, button):
post_index = self.posts_layout.indexOf(button.parentWidget())
if post_index != -1:
post = self.posts[post_index]
if self.current_user not in post["dislikes"]:
post["dislikes"].add(self.current_user)
if self.current_user in post["likes"]:
post["likes"].remove(self.current_user)
self.update_posts()
else:
QMessageBox.information(self, "Dislike", "You have already disliked this post.")
```

delete_post(): Deletes a specific post.
add_comment(): Adds a comment to a specific post.
like_post(): Likes a specific post and handles like-dislike toggling.
dislike_post(): Dislikes a specific post and handles dislike-like toggling.
Updating the UI
pythonCopy code

```
def update_posts(self):
for i in reversed(range(self.posts_layout.count())):
self.posts_layout.itemAt(i).widget().deleteLater()

for post in self.posts:
post_widget = QtWidgets.QWidget()
layout = QVBoxLayout(post_widget)

delete_button = QPushButton("Delete")
```