COMP2011

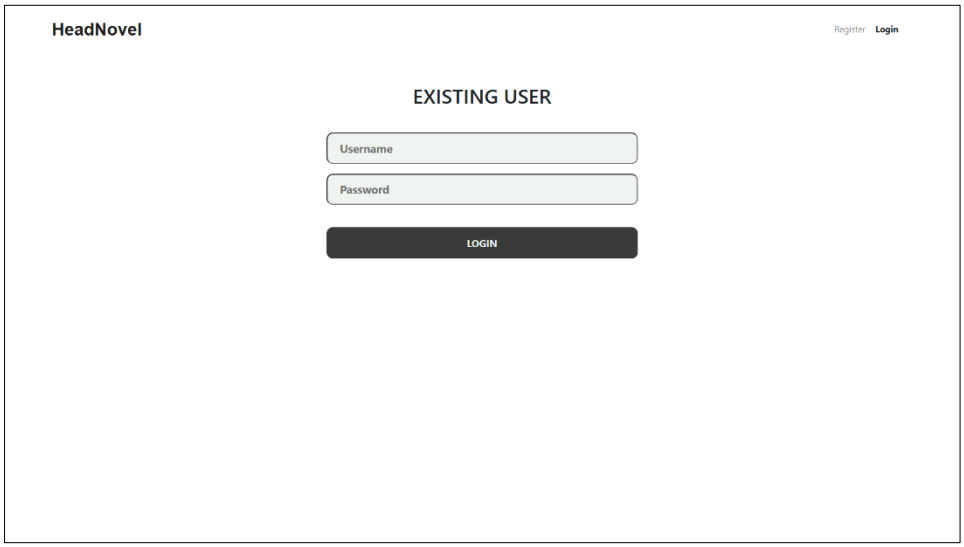
DANIAL EIZLAN BIN DAUD

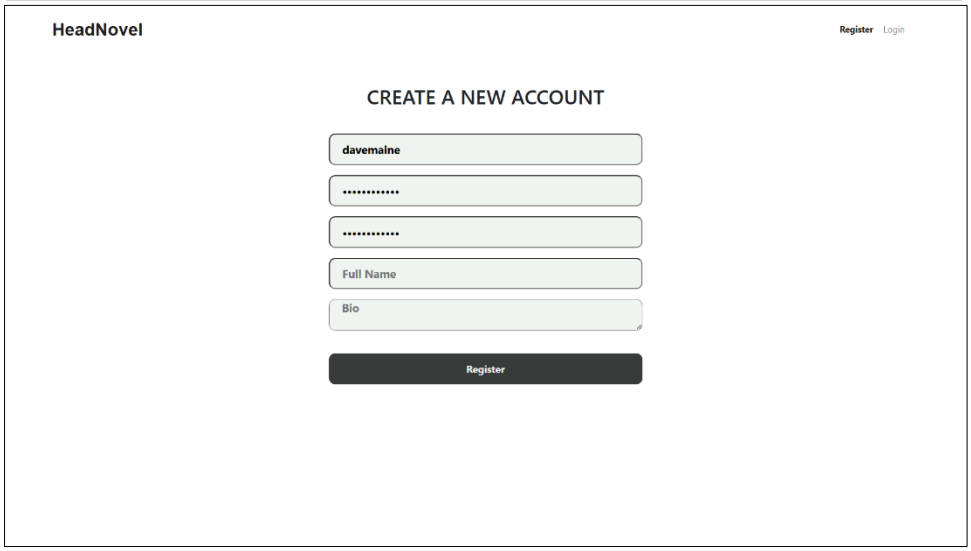
My Website

HeadNovel is a basic social media site where users can post small stories on their feed. They also may add other users as friends, allowing both parties to see and like each other's posts.

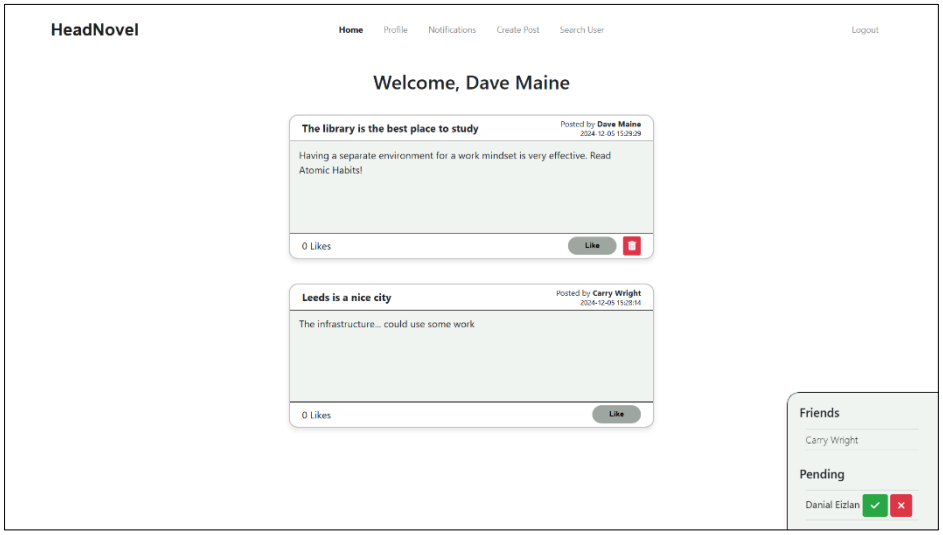
User Path

A user is greeted by the Login page. If they do not have an account, they may go to the Register page through the navigation bar to create one, after which they will be redirected back to the Login page to enter their credentials. If the user has logged in previously, they will be sent to the dashboard automatically.



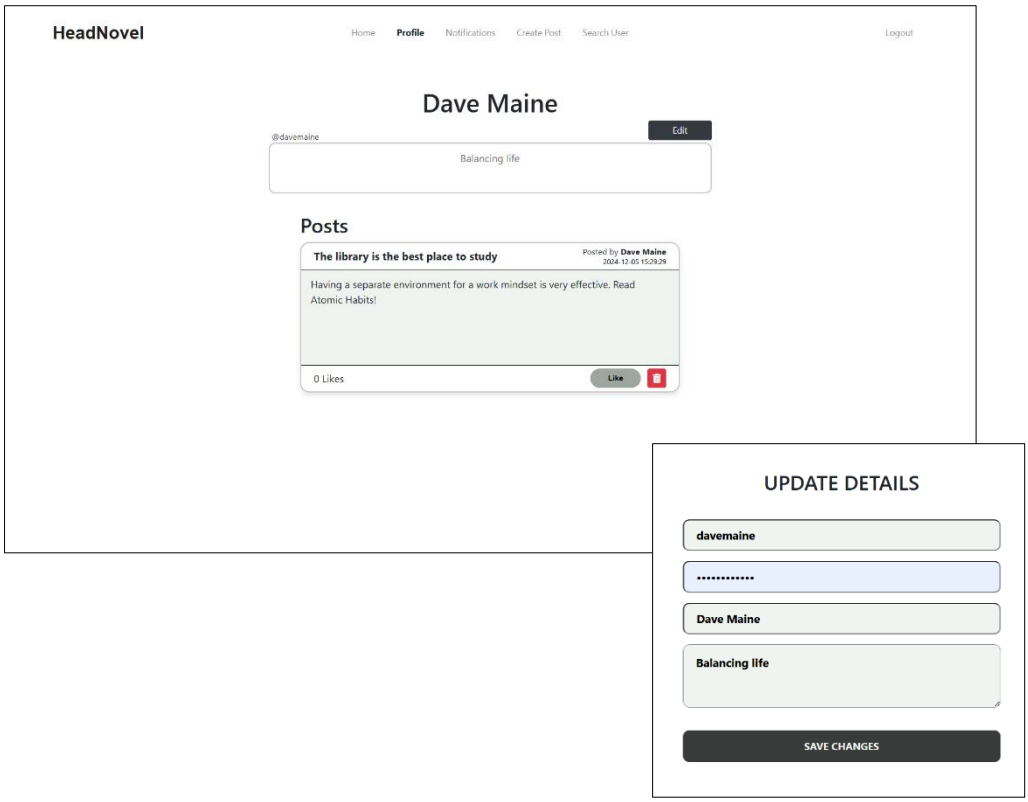


On the dashboard, the main content is topped with a greeting, followed by a series of most recent user posts. The posts displayed are limited to the ones created by the current user and their friends. Each post has a like button and if it was created by the current user, an additional delete button is visible.

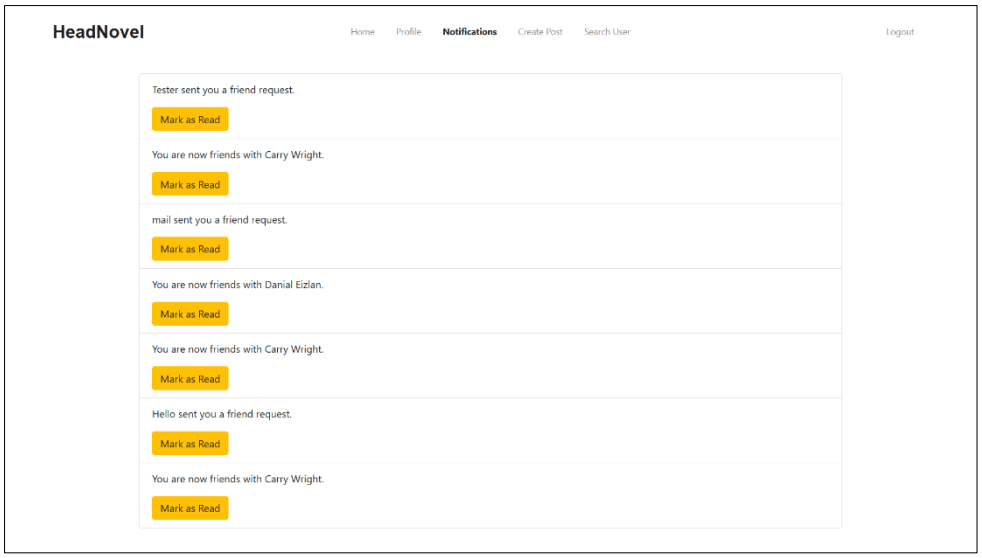


A side bar is present, showcasing the user's friends list. Any pending requests are stacked at the bottom, with the sender's name and buttons to accept or decline the friend request.

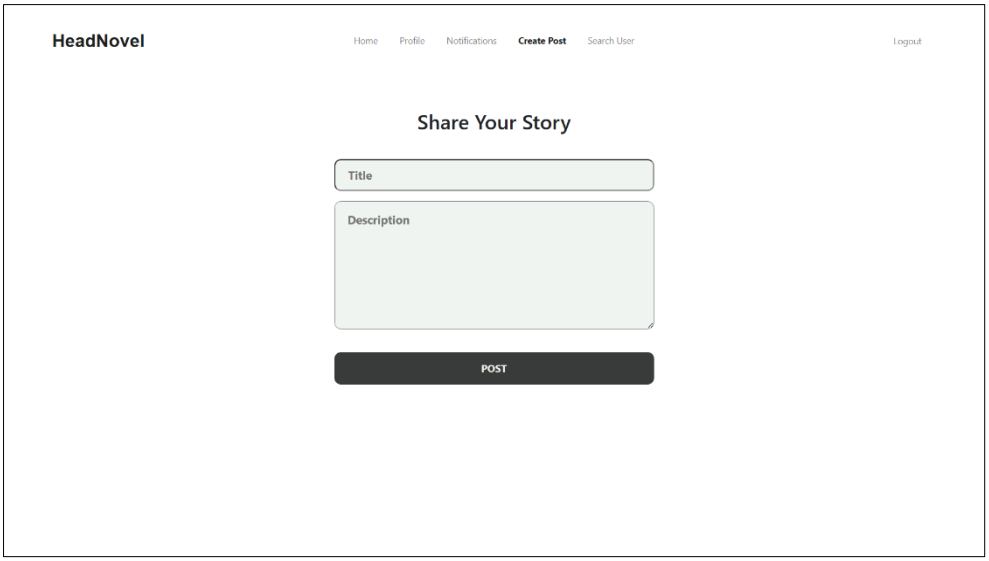
When the user is logged in, the navigation bar is changed to allow access to other pages such as the Profile, Notifications, Create Post and Search User page. On the Profile page, it shows the current user's details and posts. An edit button is available for the user to change their password, full name or biography on a separate form. If a profile viewed is not the current user's, the edit button is replaced by an Add Friend or Remove Friend button, depending on the friendship status between the two accounts.



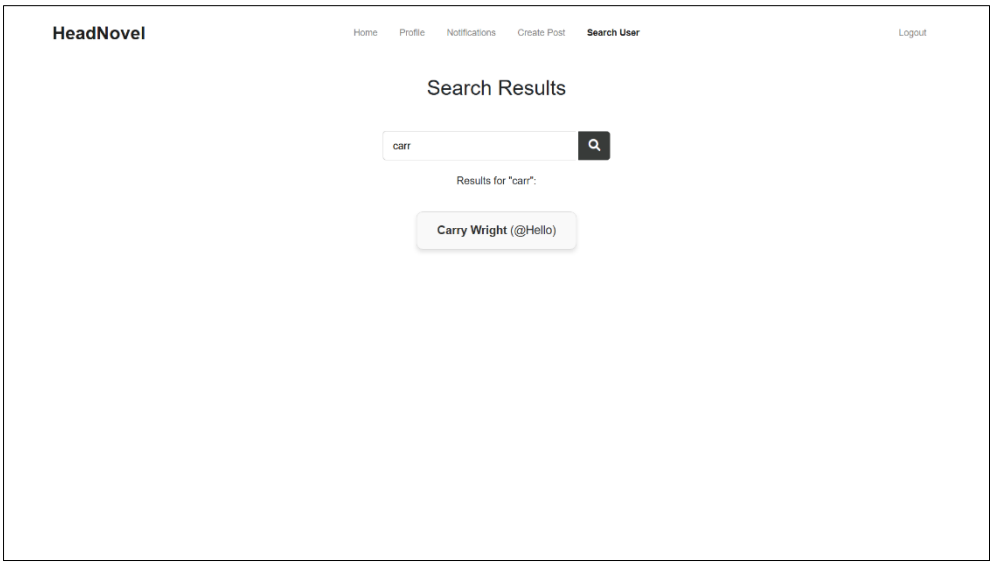
The Notifications page lists recent activities relevant to the user, including likes for their own posts and any incoming and accepted friend requests. Each can be marked as read to be removed from the list.



The Create Post page simply displays a form for the user to fill in to enter a title and description for their post before submitting.



Finally, the Search Users page allows a user to search for existing profiles, retrieving results that match a username or full name. Each result links to its respective profiles.



Design

In this coursework, I opted for a minimalistic design for a clean and user-friendly interface. First, the navigation bar provides quick access to the main pages and consistently positioned at the top.

The dashboard contains all the most relevant information for a user once they've signed in. It displays all the posts in a chronological order to allow the user to catch up on their friends' most recent updates. The sidebar is stuck to the window so the user can immediately see any pending requests available. Each post has clear sections for the title, description, like count and user information. The buttons are positioned intuitively on the bottom right corner.

The Notifications and Search User pages list their elements in an orderly manner, allowing for scalability for large datasets. Moreover, the forms are clear and simple, whether for logging in, registration, post creation or profile editing.

Database

The database for this project was designed using SQLite, with models implemented via SQLAlchemy. It has 5 key models structured to maintain relationships between users, posts, likes, friend requests and notifications.

The User table stores user details such as username, password, full_name, bio, friend_count and post_count. It has helper functions like set_password, check_password and is_friends_with to aid in authentication related functionality. The Post table stores information such title, description, likes and timestamp. The likes on each unique post are tracked through the PostLikes table. The FriendRequest table handles friend request functionality with fields for sender_id, receiver_id, status and timestamp. Its helper functions include send_request and respond_to_request to manage request status. Finally, the Notification table stores all notifications for user actions.

In the 1st coursework, helper functions were absent. Instead, I implemented most of the logic for features in views.py, which is less optimal and made the views very cluttered.

In this iteration, all the methods are refactored into each relevant model, ensuring better code readability, testing and more importantly reusability.

The relationships in this relational database can be summarized as:

User ↔ **FriendRequest** ↔ **User** (many-to-many).

User ↔ **Post** (one-to-many).

Post ↔ **PostLikes** ↔ **User** (many-to-many).

User ↔ **Notification** (one-to-many).

The many-to-many relationships in the database significantly help enhance the user experience as they allow for certain features such as:

- Displaying a real-time feed of posts from friends.
- Managing friend requests and notifications efficiently.
- Tracking and displaying user interactions (likes).

Advanced Features

Dynamic Friendship Management

This system allows users to send, accept and remove friends. It utilises the many-to-many relationship established in the database. The friend_association table facilitates by-directional friendships between users to avoid duplication through constraints. For example, the function accept_friend_request handles updating friendships, maintaining accurate friend counts and sending notifications to the users involved. The code snippet for it is shown below:

```python
def accept_friend_request(self, request_id):
    request = FriendRequest.query.get(request_id)
    if request and request.status == "pending":
        request.status = "accepted"
        sender = User.query.get(request.sender_id)

        # Handle reciprocal requests
        reciprocal = FriendRequest.query.filter_by(
            sender_id=self.id, receiver_id=sender.id, status="pending"
        ).first()
        if reciprocal:
            reciprocal.status = "accepted"

        # Update friendship and count
        if sender not in self.friends:
            self.friends.append(sender)
        if self not in sender.friends:
            sender.friends.append(self)
        sender.friend_count += 1
        self.friend_count += 1
        db.session.commit()

        # Notify both sender and receiver
        Notification.create_notification(sender.id, f"You are now friends with {self.full_name}.")
        Notification.create_notification(self.id, f"You are now friends with {sender.full_name}.")

        return "Friend request accepted"
    elif request.status != "pending":
        return "Request already accepted/declined"
    else:
        return "Invalid request or status"
```

AJAX-Based Like Button

A like button was implemented for posts. The like button triggers asynchronous requests to the server. At the same time, the button state and like count are updated instantly. The helper function in the PostLikes model manages the logic for adding and removing likes, as well as notifying the post owner when their post is liked by another user. The AJAX code snippet for this feature:

```
$(document).ready(function () {
    let csrfToken = $('meta[name="csrf-token"]').attr('content');

    $('.like-button').click(function (event) {
        event.preventDefault(); // Prevent default behavior

        let button = $(this); // Reference to the clicked button
        let post_id = button.data('post-id'); // Post ID from data attribute

        $.ajax({
            url: '/like_post/' + post_id,
            type: 'POST',
            headers: {
                'X-CSRFToken': csrfToken,
            },
            success: function (response) {
                if (response.status === 'success') {
                    // Update like count
                    let likeCount = $('#like-count-' + post_id);
                    likeCount.text(response.new_like_count); // Update the like count

                    // Toggle the button text and data-liked attribute
                    if (response.action === 'liked') {
                        button.text('Unlike');
                        button.data('liked', true);
                    } else {
                        button.text('Like');
                        button.data('liked', false);
                    }
                } else {
                    console.error('Error:', response.message); // Log any errors or messages
                }
            },
            error: function (error) {
                console.error('Error liking/unliking the post:', error);
            }
        });
    });
});
```

Real-Time Notifications

Notifications are created dynamically using the create_notification method in the Notification model. This method is automatically called whenever certain actions occur (e.g. sending a friend request or liking a post). Through database queries, they are retrieved and displayed efficiently with options to mark them as read.

Testing & Critical Analysis

Testing

Functionality

The functionalities of the application were tested extensively to ensure that all features work as intended. Below are **some** but not all the key components tested:

User Authentication

- ✓ Registering a new user with valid and invalid data
- ✓ Logging in with valid and invalid credentials
- ✓ Logging out and confirming the session data is cleared

Error messages were appropriately flashed for invalid input whereas successful registration, login and logout processes were confirmed by accessing restricted pages.

Friend Management

- ✓ Sending, accepting & declining friend requests
- ✓ Removing a friend
- ✓ Confirming that friendship status is accurately reflected for both users

The actions worked as expected, with real-time reflection on the friends list.

Post Creation & Interaction

- ✓ Creating posts with valid and invalid data
- ✓ Liking and unliking posts to check if like counts are updated correctly

Posts were successfully created and displayed on the dashboard whereas the like button is dynamically consistent.

Edge Cases Tested

- ✓ Preventing duplicate friend requests
- ✓ Liking same post multiple times
- ✓ Removing a user who is not a friend

Appropriate error messages are displayed seamlessly without system crashes.

Layout & responsiveness

Bootstrap and a mix of custom styling classes were used to ensure responsiveness to differing window sizes. Bootstrap's flex class was used on most components for proper alignment. Upon resizing the window, the navigation bar collapses nicely and the minimum properties of some components prevent shrinking past the page's limits.

Accessibility

Colour Contrast

The colour palette was carefully chosen to maintain a clean and readable interface. It ensures appropriate contrast for users with colour blindness (Cravit, 2022). The contrast ratios of text against backgrounds comply with the WCAG standards.

Keyboard Navigation

All interactive elements are selectable through the Tab key. Focus indicators are present to make keyboard navigation more intuitive.

Analysis

For each section below, try and come up with at least 2 things you can talk about. You should avoid discussing things outside of your control ('I did not have time to…', 'I could not work out how to…') and instead focus on specific issues you had. Again, you can include screenshots of your website or snippets of your code if you want to.

What Went Well

3 things that went well are the post design, login system and friend system. Taking inspiration from Facebook posts, the design was modified accordingly. Positioning the sub-elements of the post proved slightly tricky, specifically the stacking the timestamp and poster details together while maintaining the same vertical position as the title. The result is a minimalistic design that fits perfectly in my website. As for the friend system, after implementing the backend logic, most of the work was simply connecting the frontend elements to the appropriate helper functions.

What Went Badly

What did you find particularly challenging, and why? What specific problems did you have?

What was particularly challenging was the implementation of the like button. Coding the proper AJAX required both updating the frontend and backend, without reloading the page. The backend was simpler however the frontend had multiple issues. At first, the button's text content would refuse to update despite the like count being accurate after a page reload. Next, the button's text content would not reflect the status accurately upon page reload. This meant I had to also check the user's like status of a post when the page loads and modify the button's text content accordingly.

Another obstacle was styling any element that also had bootstrap. Due to the nature of bootstrap classes, they have predefined properties. The full properties of each class were hard to find online, and as a result sometimes my custom styling would not properly override the ones set by Bootstrap.

In the end, everything works perfectly which is the important result.

Improvements

If given the opportunity to restart the project, I would do some things very differently. In terms of the development approach, I would tackle the implementation of a feature one by one, rather than fully implementing the backend of the application before beginning the frontend design after. I believe this is more akin to agile development which I believe is much more efficient and will save more time.

In terms of the features, I would love to add a comment and messaging system. This will make the website much more interactable with each user's friends compared to just the viewing and liking of their friend's posts.

Bibliography

Cravit, R. 2022. How to use color blind friendly palettes in your design. *Venngage*. [Online]. [Accessed 6 December 2024]. Available from: https://venngage.com/blog/color-blind-friendly-palette/.

Divine, A. 2023. Minimalist color palette and typography in web design. *Bejamas*. [Online]. [Accessed 6 December 2024]. Available from: https://bejamas.io/blog/minimalist-color-palette-and-typography-in-web-design.