AbraCollabEra

JIB 4200

Arzeen Ahmed, Crystal Escoda, Paul Hutchison, Allen Mai, Anya Sharif

Client: Akosua Ajabu

Repository: https://github.com/pbhutchison/akosua-artistry-JIB4200

Table of Contents

Table of Figures

Terminology

API (Application Programming Interface): A set of rules that allows different software applications to communicate and interact with each other.

Backend: The part of a website or application that operates behind the scenes, handling data processing, storage, and server-side logic.

CSS (Cascading Style Sheets): A language used to style and layout web pages, including design elements like colors, fonts, and spacing.

Database: A structured system for storing and managing data for easy access and manipulation.

Flask: A lightweight Python web framework for building simple, flexible web applications.

Frontend: Frontend refers to the visual and interactive part of a website or app that users see and engage with directly.

JSON (JavaScript Object Notation): A lightweight data format used to store and exchange data in a readable and easy-to-parse format, often used in web APIs and configuration files.

Modal: A component that overlays other content on the webpage. These UI elements are used to provide additional content or prompt user interaction related to the main page while keeping information organized.

MySQL: An open-source database system that stores data in tables and uses SQL for querying.

React: A JavaScript library for building fast, dynamic user interfaces with reusable components.

SQL: A standardized language used to manage and manipulate databases by performing tasks like querying, updating, and organizing data.

Introduction

Background

Our product, AbraCollabEra, is a cross-platform website built using ReactJS for front-end, Python Flask for back-end, and MySQL for storing user specific data. Our goal for this website is to provide beauty industry professionals with a reliable and easy-to-use platform where they can discover potential colleagues, coordinate and schedule events, access educational content, and show off their own work. Our website includes a social feed, messaging features, interactive courses with videos that users can watch on-site, and personalized calendars. Users are also able to filter search their network to find other users they want to connect with. The purpose of this document is to provide details of the architecture and design process of AbraCollabEra.

Document Summary

The System Architecture section provides a high-level overview of how the major components of our system interact with each other, using two diagrams. The static system architecture diagram provides insight into how the website is organized. The dynamic system architecture diagram displays a step-by-step example scenario of how the application performs its tasks.

The Component Design section outlines the components of our system architecture in more detail, giving a more detailed insight into the system. We will "zoom-in" to view the system on the component level, accompanied by descriptions and diagrams of the static relationships and dynamic interactions between components.

The Data Storage Design section describes how our system's database structure works, which consists of the MySQL database. File use, data exchange, and security concerns are addressed in this section as well.

The UI Design section presents the primary user interface screens, functioning as a walkthrough of how users interact with the website, backend, and device.

System Architecture

Introduction

Our application, AbraCollabEra, is a web-based platform designed to run seamlessly on both desktop and mobile devices. It aims to provide a secure and collaborative environment where users can interact, share content, and manage their schedules effectively. The platform is built to prioritize usability and scalability while maintaining a few safety features to protect user data.

We chose a layered architecture consisting of multiple layers:

- **Presentation Layer (React Front-End)**: handles user interactions and web rendering of the web interface allowing users to access various features.
- **Business Layer (Flask Back-End)**: manages the application's logic, processing any API requests and facilitates communication between the front-end and data layer.
- **Persistence Layer:** represents all the key entities of the system.
- **Database Layer (MySQL Database & Static Storage):** provides storage for both structured and unstructured data.

This architecture allows us to develop and test each layer independently, offering flexibility and reducing the risk of interdependencies. It also simplifies maintenance by isolating concerns, so updates or changes to one layer do not impact others.

In the following section, we present an overarching view of our system to provide a better understanding of what happens behind the scenes. Additionally, we showcase an example of a common task a user might perform while using our application.

Rationale

We chose a layered architecture for our web application because it allows us to organize the system into distinct layers: the React Front-End, the Flask Back-End, and the MySQL database. This separation enables independent development and testing of each layer, improving flexibility and efficiency in our development process. By isolating concerns, we ensure that changes or updates in one layer do not impact the others, saving time and effort in maintaining and scaling the system.

This architectural approach also helps us address critical security issues that were essential to meeting the client's requirements. For instance, we implemented user authentication at the API level using Flask, employing JSON Web Tokens (JWT) for secure session management. In addition, the front-end is responsible for validating user input to reduce risks such as injection attacks, while the back end performs secure encryption before storing user passwords. By dividing responsibilities across these layers, we ensure that security and functionality are handled effectively and that potential vulnerabilities are mitigated at multiple levels of the system.

Static System Architecture

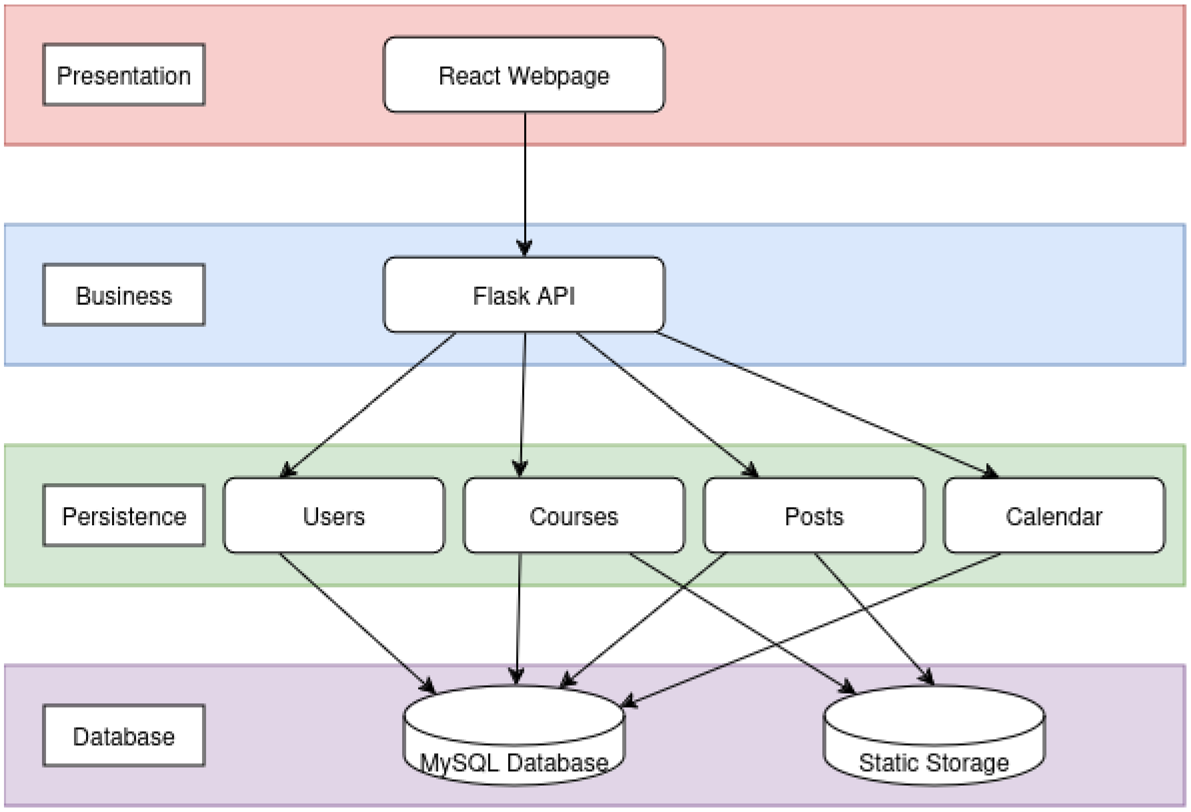


Figure 1: Static System Architecture Diagram

In Figure 1, our Static System Architecture Diagram showcases the design of our application. The first layer, the **Presentation Layer**, represents what users see and interact with, supported by our front-end technology, React. This layer provides a flawless and dynamic user experience.

The second layer, the **Business Layer**, is implemented using the Flask API. It handles all of the application logic, processes user requests, and serves as a bridge between the front-end and the data layer.

The third layer, the **Persistence Layer**, organizes the important entities of our application, such as users, courses, posts, and the calendar data. This layer ensures efficient data management and logical relationships among these components.

Finally, the Database Layer includes a MySQL database, which securely stores user information, and a static storage system, which handles media content uploaded by users. This separation of structured and unstructured data supports efficient data operations and scalability.

This system architecture allows independent development and testing of each component, ensuring modularity and maintenance. By isolating any concerns, the architecture makes it easy to flawlessly integrate and interact between layers, making the system scalable for future enhancements.

Dynamic System Architecture

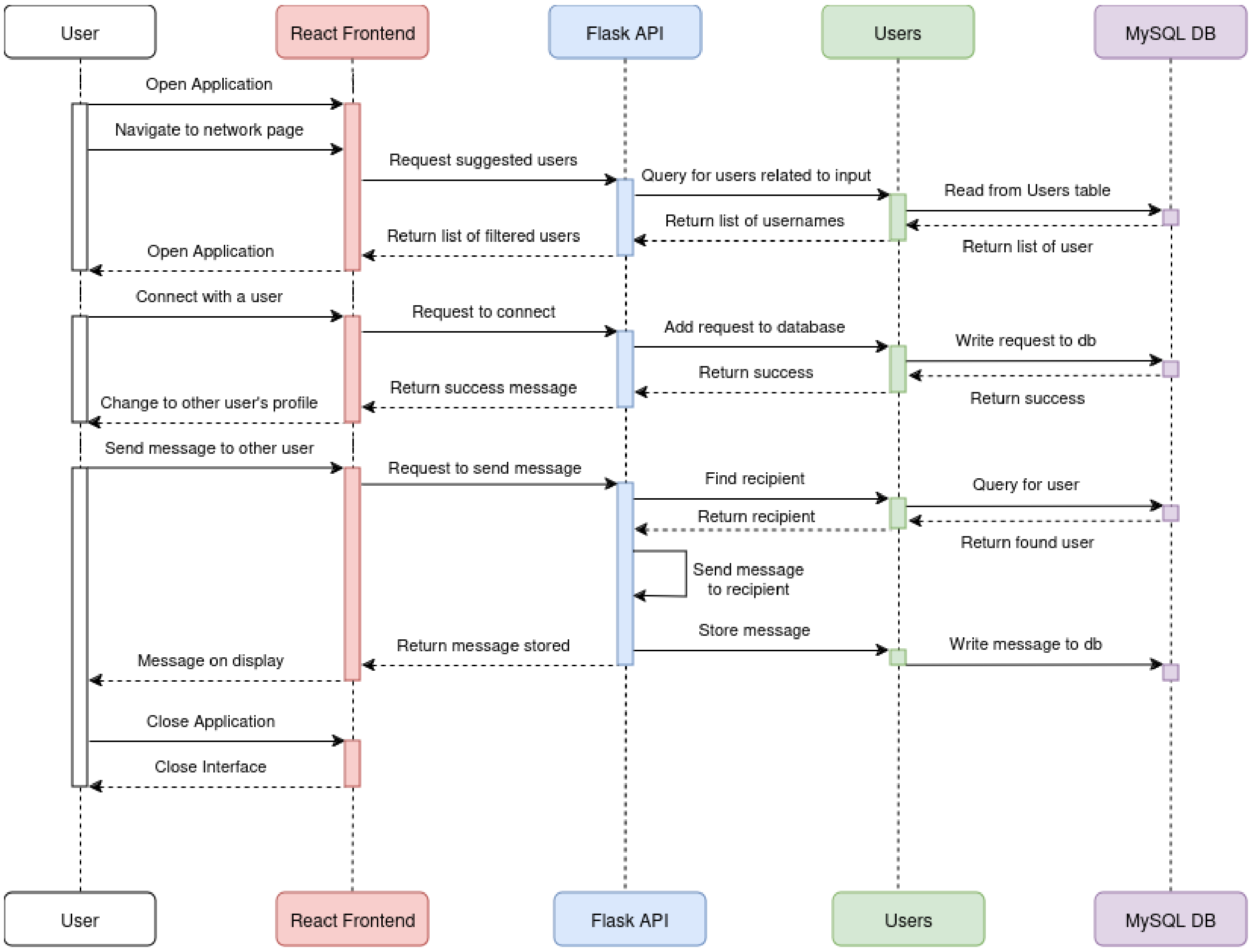


Figure 2: Dynamic System Architecture Diagram

In Figure 2, our Dynamic System Architecture Diagram illustrates the interaction between various components as the user completes a specific task. In this example, the user browses through a list of other users to establish a connection and send a message to their selected recipient.

The process begins when the user navigates to the Connect Page in the React application. At this stage, the React Front-End sends a request to the Flask API, which retrieves a list of suggested users by querying the MySQL database. The database processes the query by reading from the “Users” table and returns a filtered list to the front-end, which displays it to the user.

Once the user selects someone to connect with, the front-end sends the connection request to the Flask API, which processes the request and updates the “Connections” data in the database. After the connection is successfully added, the user can initiate a message. The message is sent to the Flask API, which stores it in the database under the “Group-Messages” table. This stored message is then retrieved and displayed on both users’ interfaces through the React front-end/.

We chose to showcase the User Connection and Messaging Feature because it demonstrates interactions across all main components of our system: the React front-end, Flask API, MySQL database, and the key entities such as users and messages. This example provides a clear view of how the different layers of our application work together.

In addition, we used a UML System Sequence Diagram to represent this dynamic architecture. This diagram style was selected for its clarity and ability to highlight the flow of interactions between the user, front-end, back-end, and database. By focusing on the sequence of operations, it provides an easy-to-follow example of our system's runtime behavior, making it a useful tool for understanding the structure and interactions within our application.

Component Design

Introduction

In the following section, we demonstrate the structure of our web application's static components and dynamic behavior of our posting feature. Figure 3 presents a Class Diagram to illustrate the static architecture of the system. This diagram was chosen because it successfully represents all the static features of our application, including attributes, methods, and the direct relationships and dependencies between components. By using a class diagram, we highlight the conceptual integrity of the system and ensure that low-level components and their roles are clearly defined.

Figure 4 is an Interaction Overview Diagram (IOD) that represents the dynamic behavior of the system when a user creates a post and submits it to the homepage feed. The IOD captures the sequence of interactions between components during the posting process, showcasing how user actions trigger changes in the application. This includes steps like user input from the front-end, data validation and processing in the back-end, and storage in the database. The use of an IOD allows us to break down complicated sequences and conditional interactions into a simple and easy-to-follow format. This provides clarity about the flow of control and data within the system.

The IOD was chosen because it visually represents the flow of interactions, including loops, decision points, and etc. This is particularly useful for designing and troubleshooting, as it provides a clear understanding of how user interactions affect different components of the systm. By underlining the dynamic interaction between the React front-end, Flask API, and MySQL database, we make sure that stakeholders can understand the functional dynamics of the posting process.

By combining both diagrams, we provide an overview of how the system operates. The Class Diagram makes sure we have a detailed understanding of the static structure, while the IOD emphasizes the runtime interactions that connect these components. Together, they show the conceptual integrity of the system and the rationale behind our design choices, connecting the static and dynamic components of the application.

Static Component Diagram

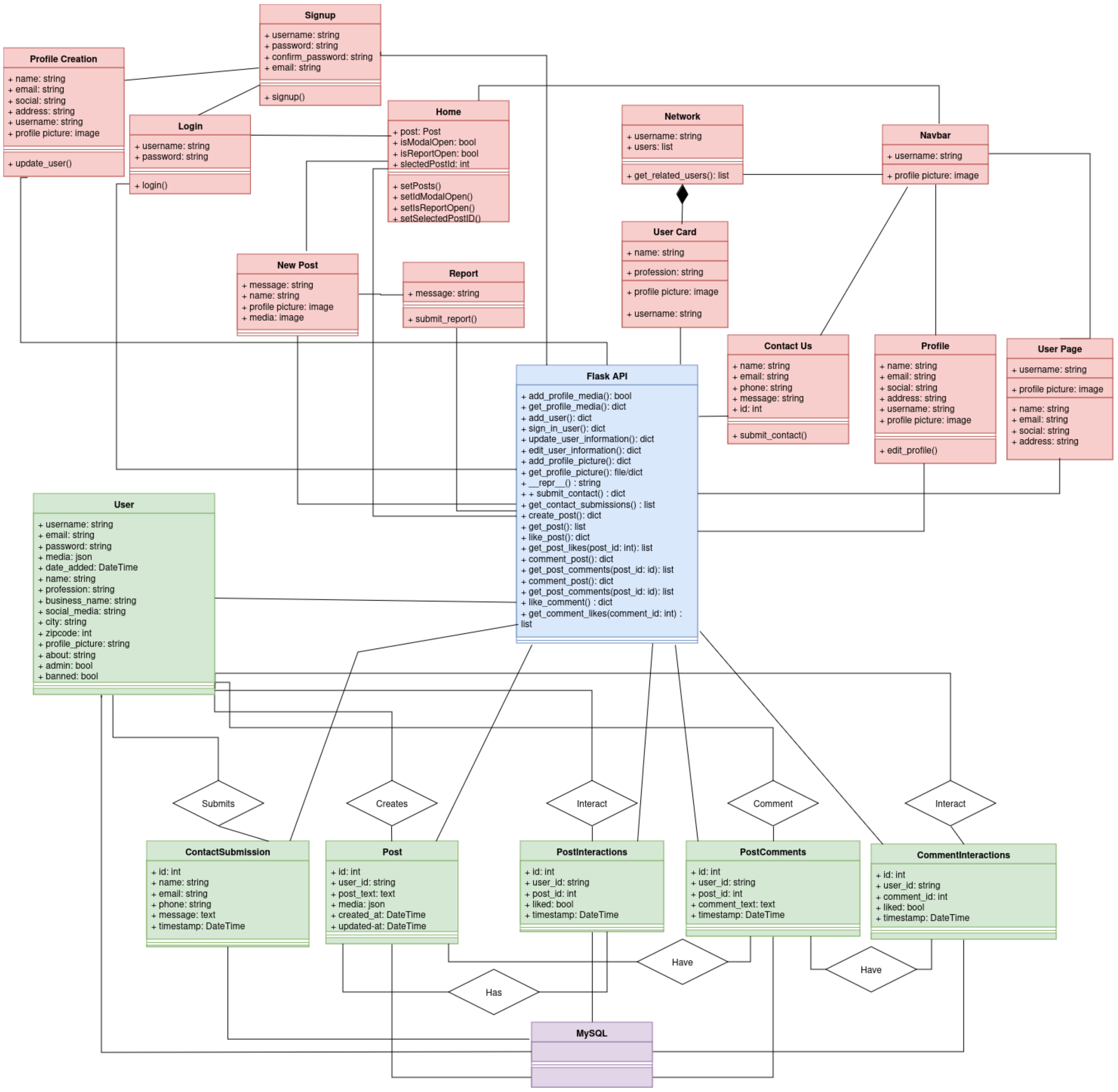


Figure 3: Static Component Design Diagram

Figure 3 above displays a class diagram that visualizes the components of our application, their relationship to other components, and the attributes and operations that they exhibit. It showcases the UI components, API, and data structure elements that interact with the MySQL database.

Our front-end components include the various pages of our web application. These include the Signup, Profile Creation, and Login pages, along with the navigation elements which include the Home, Network, Contact Us, and Profile pages.

The User class outlines the many attributes that a user of the application possesses, which includes their username, email, password, any media they upload, their name, their profession, social media links, business name, their city and zip code, their profile picture, and a description of them. These attributes are displayed throughout the various front-end components. The User class also includes Boolean values that label whether the user is an admin or is banned. These attributes help us achieve a secure platform where users can be admins that regulate banned posts, and users that are banned have restricted access and functionality in our application. The User class gets its information from the Profile Creation component when the user signs up.

The Home class shows the feed page and the new posts that users create along with existing posts posted by other users. The New Post and Post classes stores the attributes of each post, which includes the ID, the user’s name, the post’s text, the media they decide to post (optional to the user’s desire) and the date and times of the post for the posts on the feed page itself. The PostInteractions, PostComments, and CommentInteractions classes are interactions that users can do with the posts. Along with the user interactions on the post, the user can also choose to report the post for inappropriate content, whose mechanism is provided by the Report class. If users decide to further want to submit a claim about the website not necessarily related to inappropriate posts, they can do so through the Contact Us page. The Contact Us interface allows users to communicate with platform support to enhance user engagement and platform management.

The Service/API class acts as an intermediary for the data transactions between the frontend and database of our application. It includes attributes that include profile media, user information, contact information, post information like loves and comments. This class ensures data processing across the platform.

The underlying MySQL structure is detailed through tables like Post, Comment, and various interaction tables, which store and manage the extensive data generated by user activities. This setup not only supports the platform's dynamic content but also organizes data in a manner that supports quick retrieval and scalable storage solutions.

Dynamic Component Design

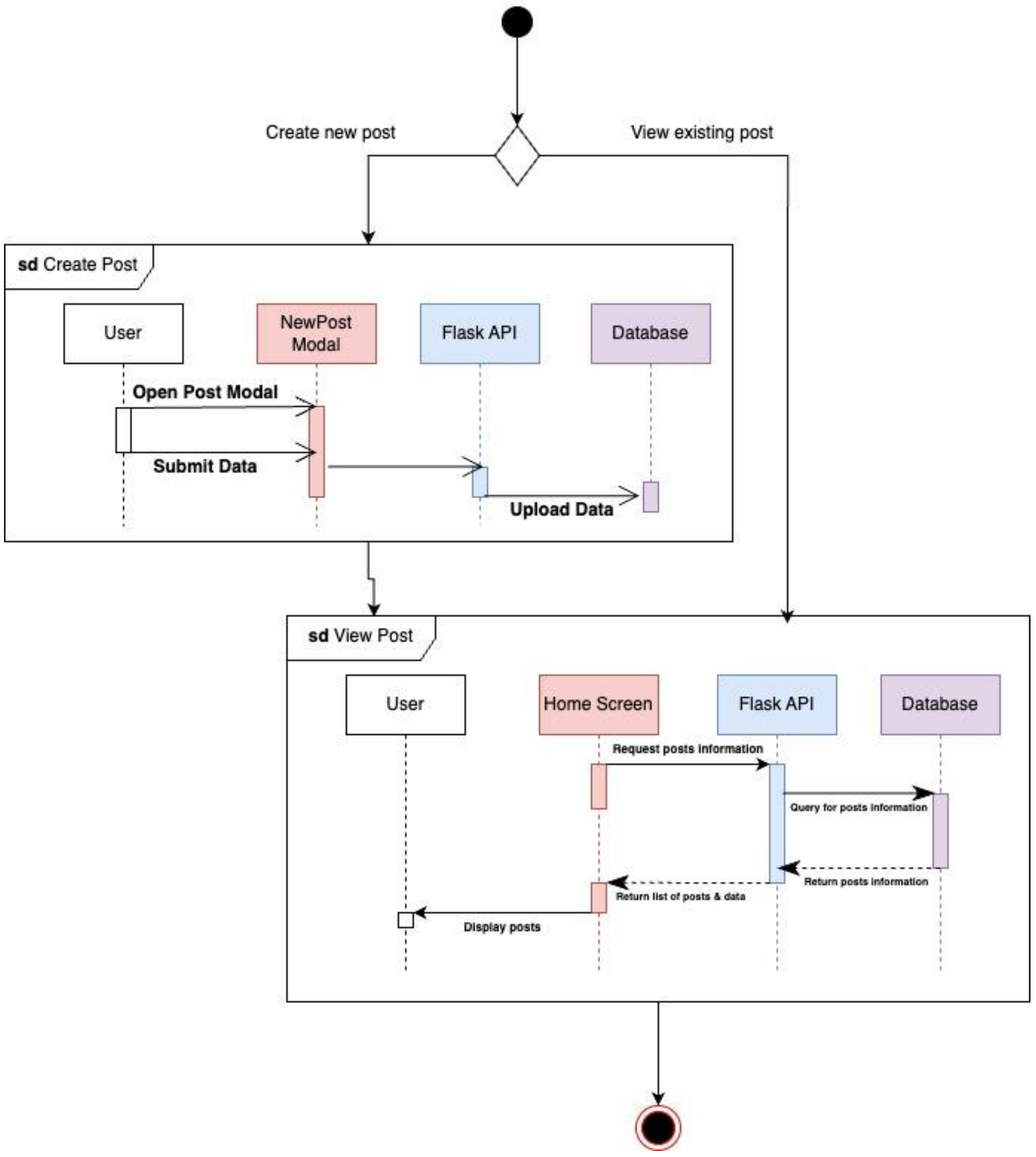


Figure 4: Dynamic Component Design Diagram

Figure 4 above illustrates an interaction flow of the user using an Interaction Over Diagram. Specifically, the diagram demonstrates a user workflow of creating and viewing posts on the home feed page of our application. When on the home feed page, a user can view existing posts or create a new post of their own. If a user wants to create a post, they click on the feed input box and are prompted with a modal in which they can input post data including, but not limited to, text input, images, videos, and GIFS. Upon entering their desired content into the modal, the user submits the data, which triggers a sequence of interactions with the back end of the application. The submission initiates a request to the Flask API. This API call involves sending the user's data to the server, where it is subsequently processed and stored in the database. The diagram highlights the flow from the user's action through the system's components up to the point of data storage and the data handling process.

Once the data is successfully uploaded to the database, the user can either continue to interact with other functionalities of the application or return to the home feed. If choosing to view posts, the user interacts with the "View Post" sequence, which starts with a request from the home screen to fetch posts. This request is processed by the Flask API, which queries the database for existing posts. The results are then returned to the application and displayed on the user's screen, allowing them to browse through various posts, including the one they might have just created. Afterwards, the user may choose to create another post or again view the current posts on the home screen.

Data Storage Design

Introduction

In this section, we will explore how data is stored in our application. First, we will look at the database schema for the MySQL database. We will also discuss how file management occurs, as well as the security measures considered when we designed the process for storing and transferring data.

Database Use

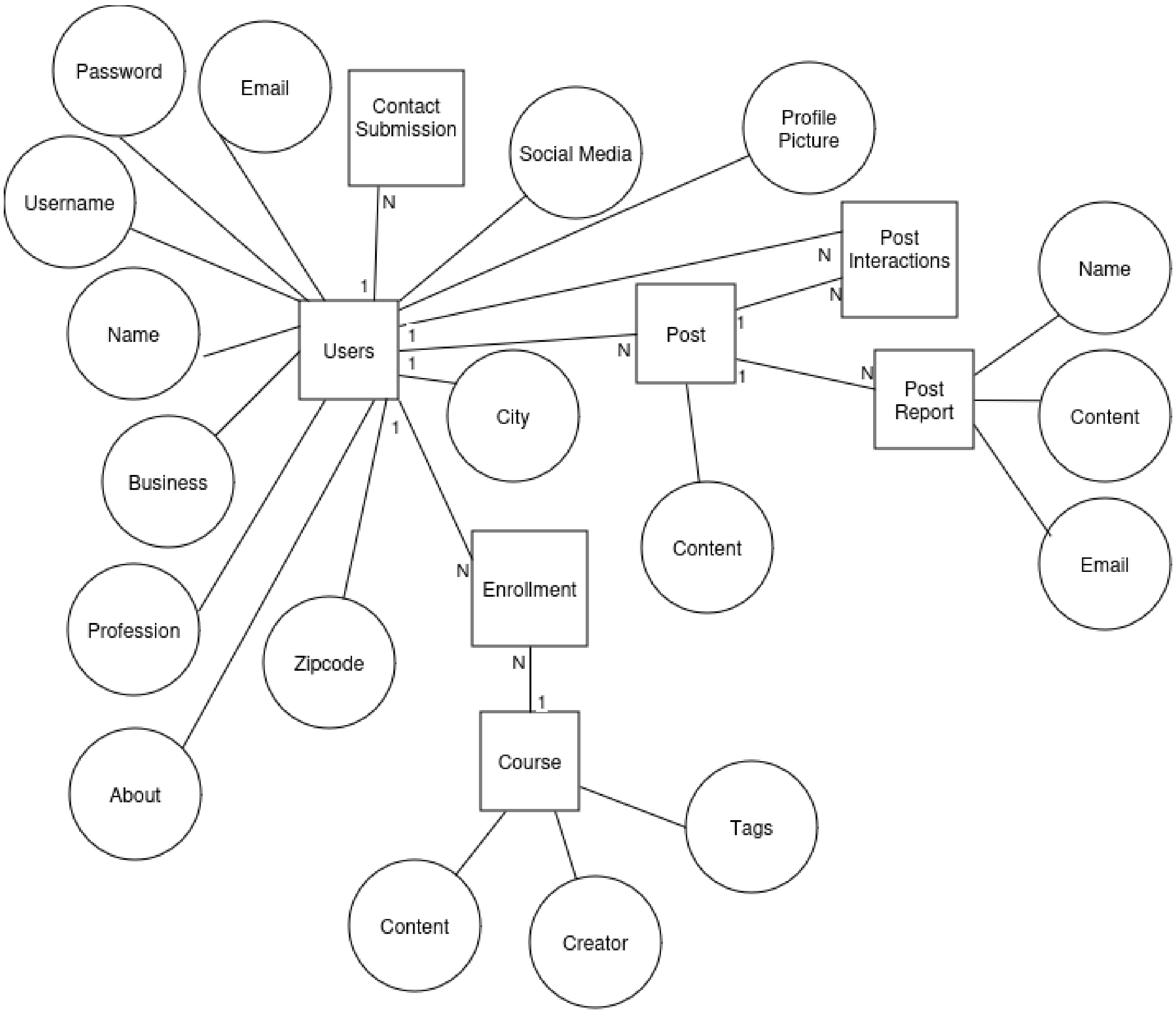


Figure 5: Entity Relationship Diagram

The Users object represents the users in our system and stores key information such as username, email, passwords, admin status, etc. This object is created when a new user registers on the website via the sign-up page. Once a user submits their data through the React.js frontend, it is processed by the Flask backend and saved in the MySQL database. Users can have various roles, such as admin or course creator, and their information is securely stored and retrieved during authentication and other interactions. They can also create/enroll in courses and make/interact with post.

The Courses object represents the courses created by users on the site, with fields like content, enrollment, creator, and associated tags. Users can create, view, and enroll in courses through the front-end, with the data being processed in the backend and stored in MySQL. Each course has a creator and tracks the number of enrolled students in an enrollment table. Courses are also linked with multiple tags, allowing users to filter/search courses easily. This object forms relationships between users and tags to ensure a structured, searchable course catalog.

The Post object allows users to create posts on the homepage. Each post contains content, the name of the creator, and can store interactions such as likes and comments. These interactions are stored in the backend and tracked in MySQL. Users can interact with posts through likes, comments which are saved in PostInteractions object, or even reporting inappropriate content via the PostReport object. Reports are stored and flagged for admin review, allowing any necessary actions to be taken. The relationships between posts, users, and interactions help maintain structured data and enable efficient post management.

Another object is contact submission where users can go to our contact page and interact with the form where fill their personal info such as name, email, and phone number along with their message that they would like to send to the website, that gets processed through our backend and saved in MySQL database.

File Use

Users can upload image and video files in standard formats (png, jpg, jpeg, mp4, etc.). Files are uploaded and given unique identifiers to be stored in a static storage directory, where they can later be retrieved and displayed on the webpage by making use of Cross-Origin Resource Sharing (CORS).

Data Exchange

All exchange of data is between the Flask API and the React webpage, via HTTP requests. This is accomplished by defining web routes in the Flask API that are invoked to request or change data in the database.

Security Concerns

While designing the database, we took security into account to ensure that users' personally identifiable information is protected. First, all data exchange between the two applications occurs via HTTPs to ensure that all the data is encrypted during transfer. We have also required that users be logged into the application to view any data within it, which ensures that all users viewing the page content are known to the system. We have accomplished this by making the routes to these pages protected from users that are not logged in. A user's password is salted and hashed when it is stored in the database using the *bcrypt* encryption algorithm. This ensures that in the case of a data breach, user accounts should not be compromised. Other personally identifiable information is not protected in the database, since it is viewable on their profile. Protecting this information in the database would not obscure their identity.

UI Design

Introduction

In this section, we present the User Interface (UI) of our multi-functional web application. As mentioned before, the site serves as a central hub for beauty professionals to access numerous resources pertaining to their professional needs. This includes networking with others in the beauty industry, accessing a catered feed of user posts, creating and enrolling in courses, and showing off their business/services. We will discuss the design decisions we made in our interface and showcase all the main screens that our users will interact with.

Login / Sign-Up

When a user first opens the application, they are prompted to enter their username and password to login. These screens are split into the interactive half, with the text fields where information is entered, and the image half, where a carousel cycles through relevant images to encapsulate the feeling we want users to experience from the site.

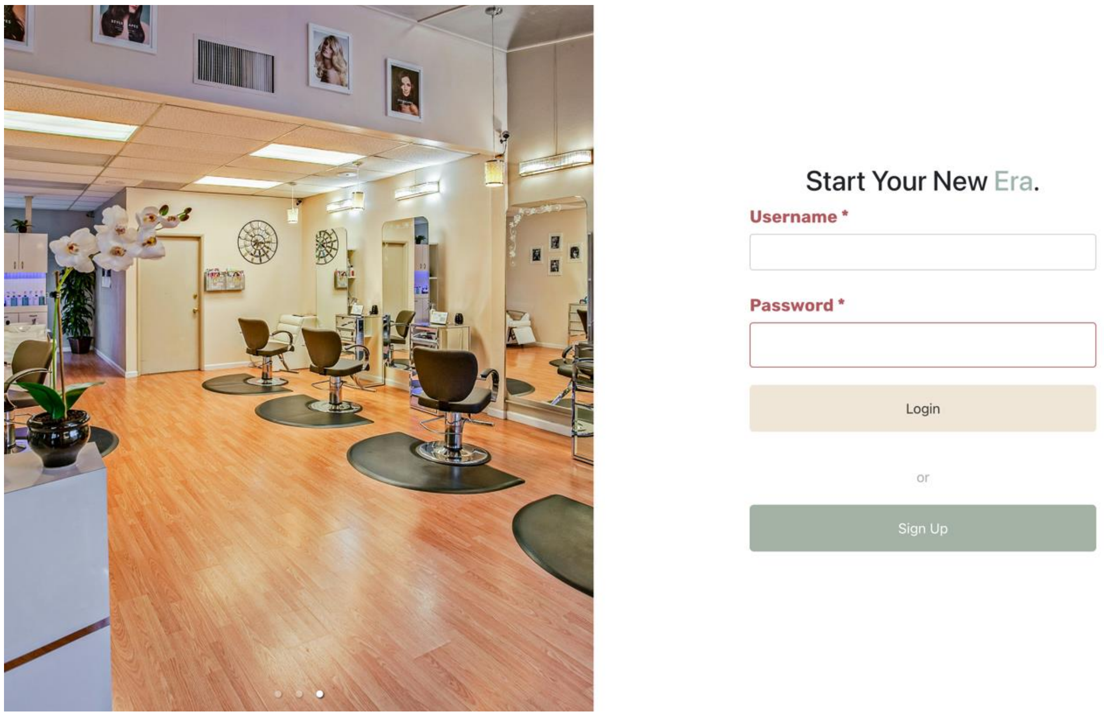
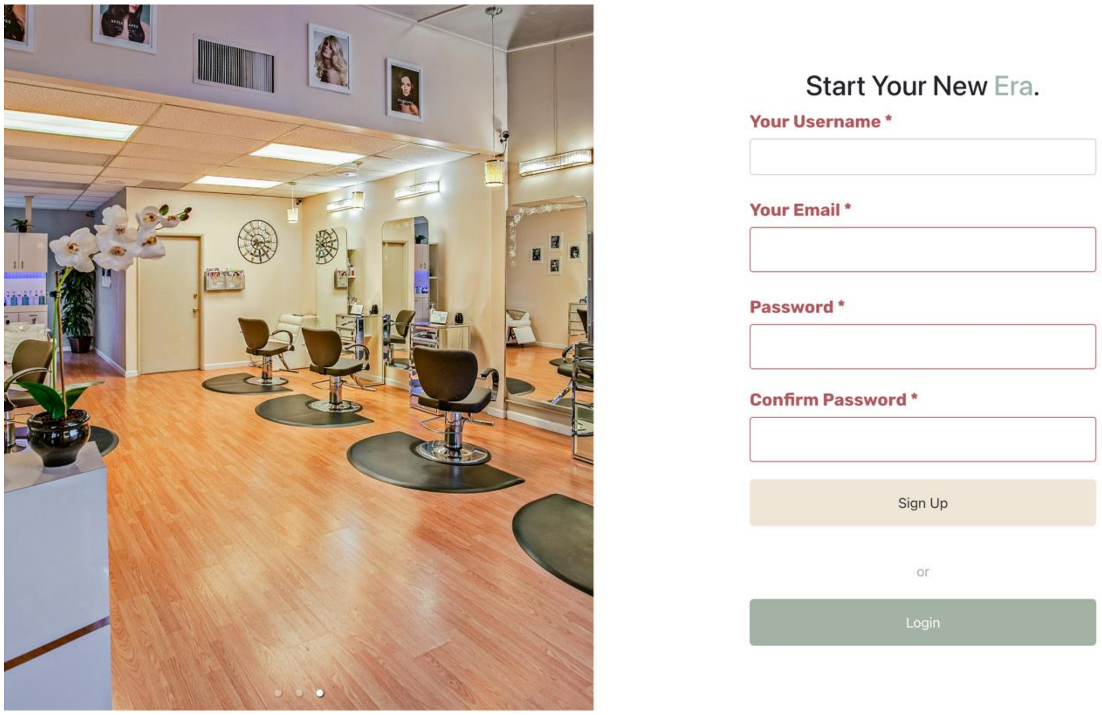


Figure 6: Login

First-time users can navigate to the Sign-Up page. We chose to make the Login and Sign Up buttons two different colors, to distinguish the two.



Start Your New Era.

Your Username *

Your Email *

Password *

Confirm Password *

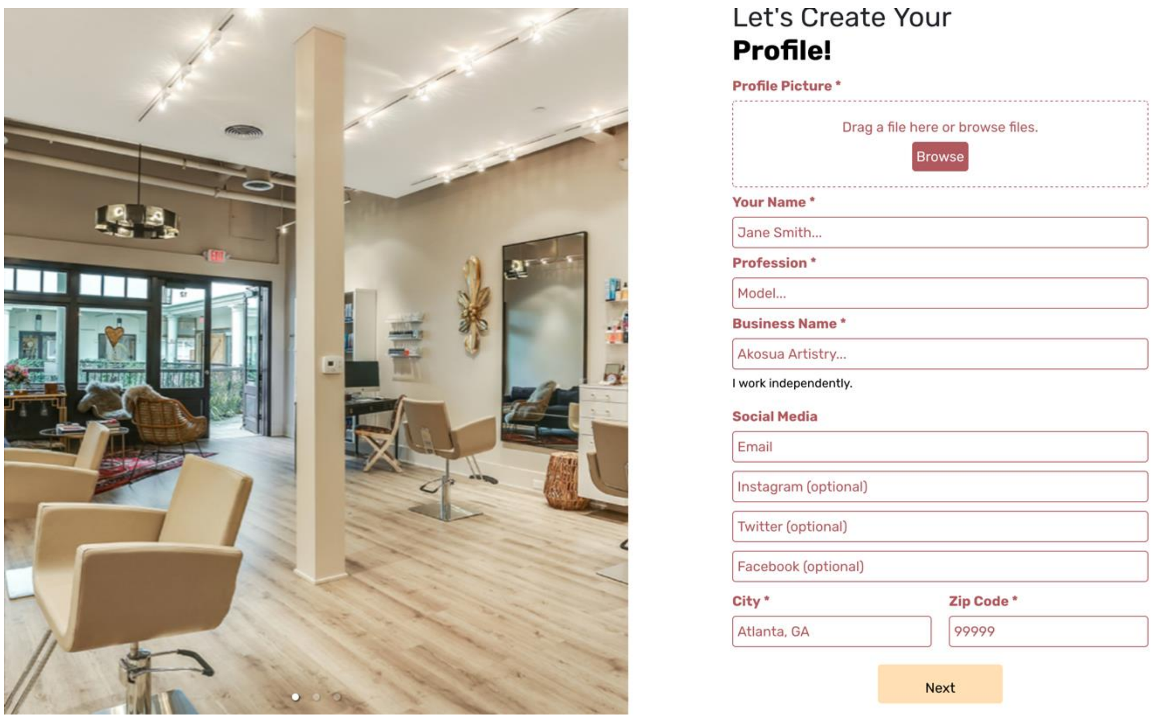
Sign Up

or

Login

Figure 7: Sign Up

Once a user enters their basic credentials, they are taken to the Profile Creation page, where they can enter their profile information, including profile picture, name, and professional details.



Let's Create Your
Profile!

Profile Picture *

Drag a file here or browse files.

Browse

Your Name *

Jane Smith...

Profession *

Model...

Business Name *

Akosua Artistry...

I work independently.

Social Media

Email

Instagram (optional)

Twitter (optional)

Facebook (optional)

City *

Atlanta, GA

Zip Code *

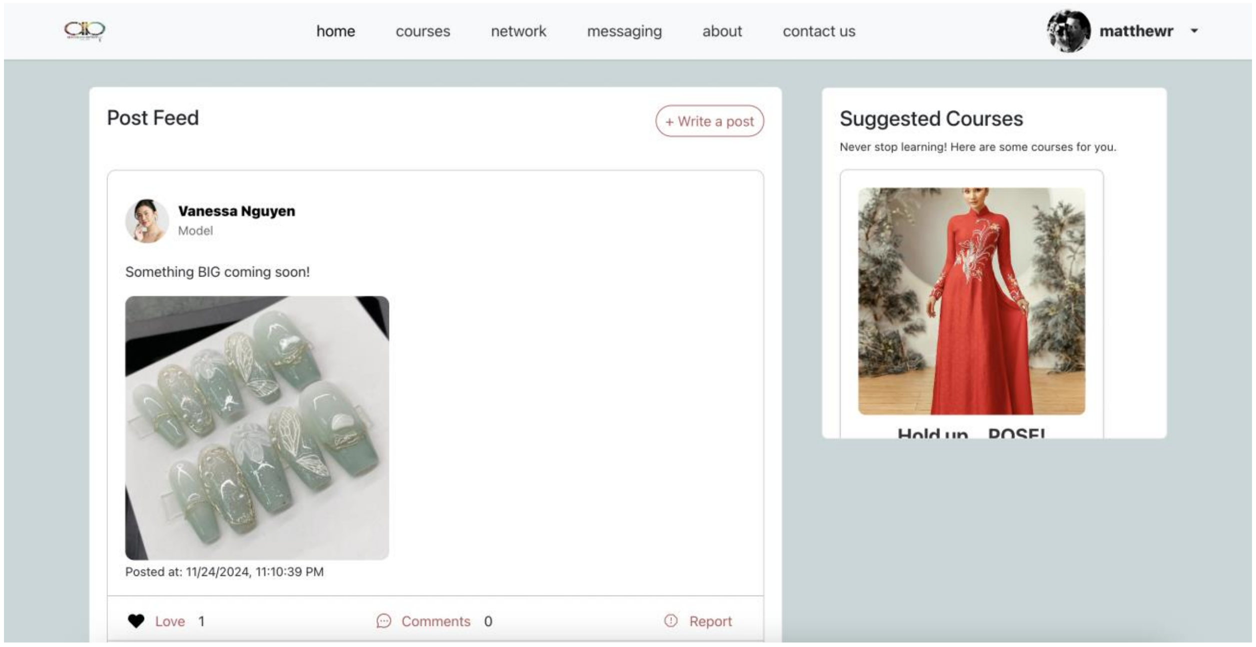
99999

Next

Figure 8: Profile Creation

Home Feed

After logging in, users are directed to the home page. This page displays a feed of all user posts, which updates in real time. Users can like, comment, or report posts. On the right, there is a section to display suggested courses, which displays a list of courses that might potentially catch the user's eye.



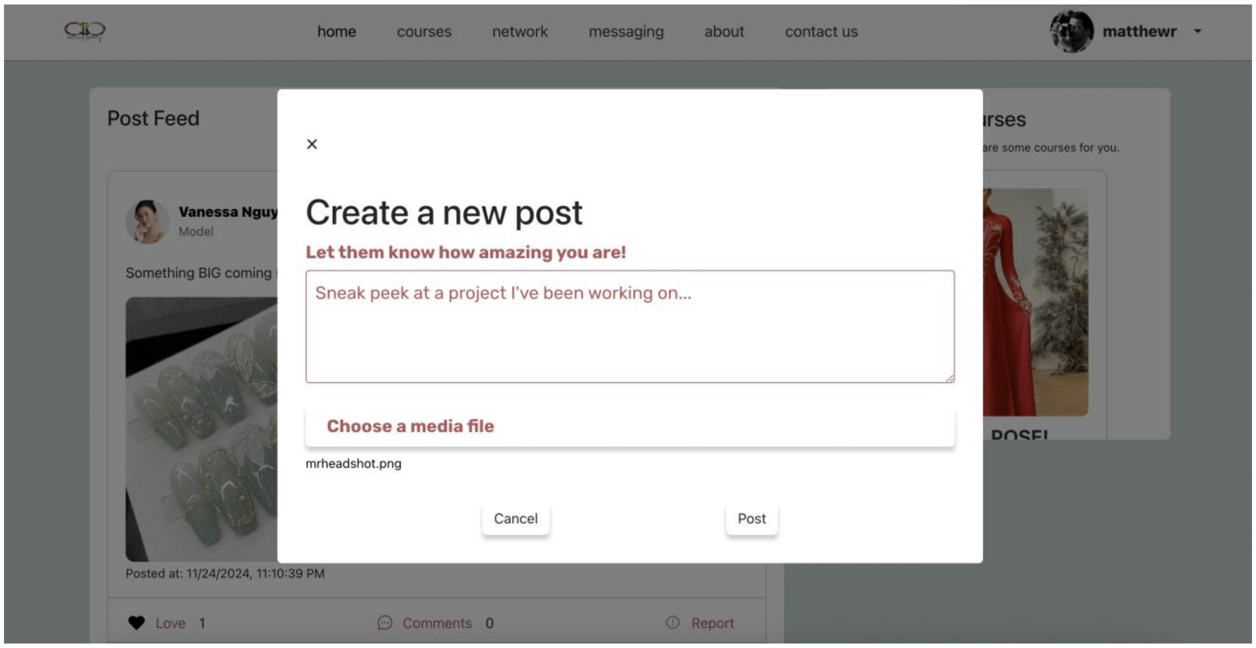
Something BIG coming soon!

Posted at: 11/24/2024, 11:10:39 PM

Never stop learning! Here are some courses for you.

Figure 9: Home Feed

Users can click the Write Post button to open a modal, where they can create new posts.



Create a new post

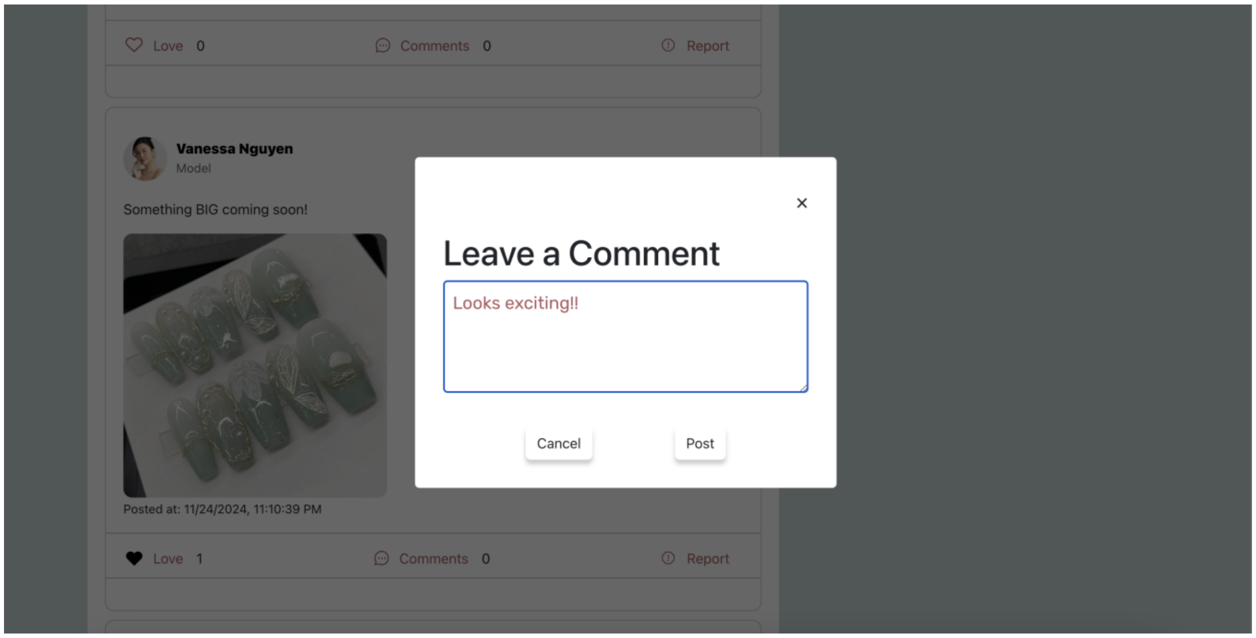
Let them know how amazing you are!

Sneak peek at a project I've been working on...

mrheadshot.png

Figure 10: Write Post

Users can also like, comment, or report posts. Clicking on Comment or Report is functionally identical to the Write Post button; on click, these buttons open a modal where users can type the response they want, then submit.

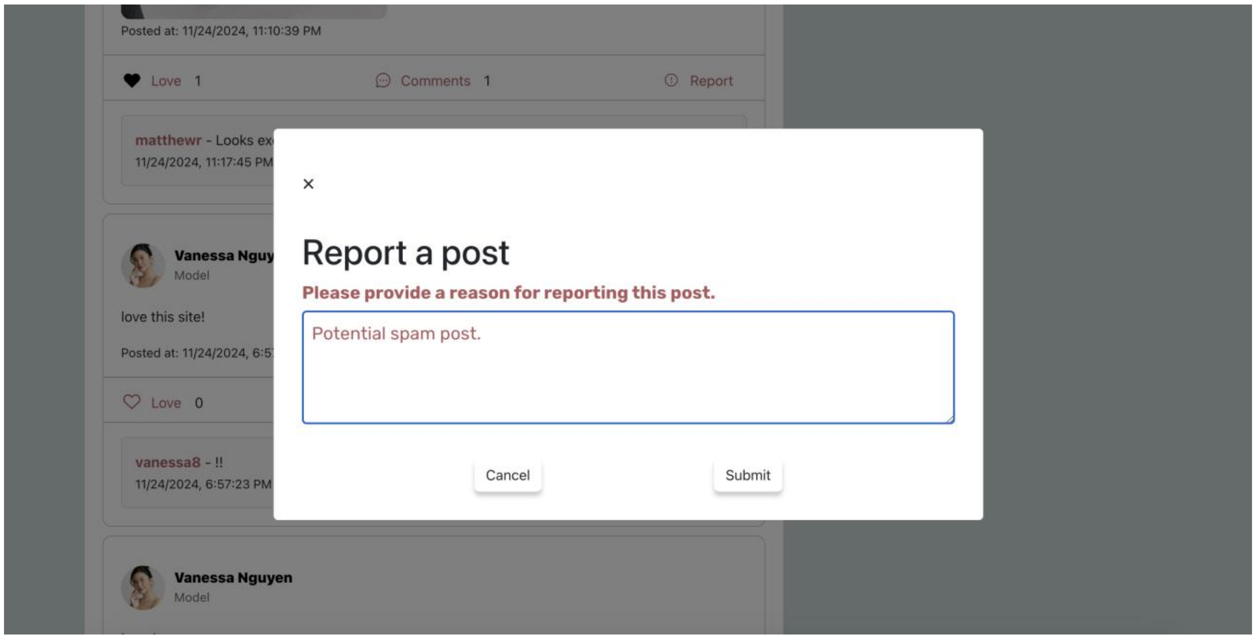


Leave a Comment

Looks exciting!!

Figure 11: Write Comment

Reports can be made by any user on any post.


Posted at: 11/24/2024, 11:10:39 PM

matthewr - Looks ex
11/24/2024, 11:17:45 PM

Vanessa Nguy
Model

love this site!

Posted at: 11/24/2024, 6:5

vanessa8 - !!
11/24/2024, 6:57:23 PM

Vanessa Nguyen
Model

Report a post

Please provide a reason for reporting this post.

Potential spam post.

Figure 12: Write Report

Courses

On this page, users can toggle between all available courses, courses created by themselves, and the courses they have enrolled in. On the Available Courses toggle, users can filter and search through all the available courses on the website to find one they want to enroll in.

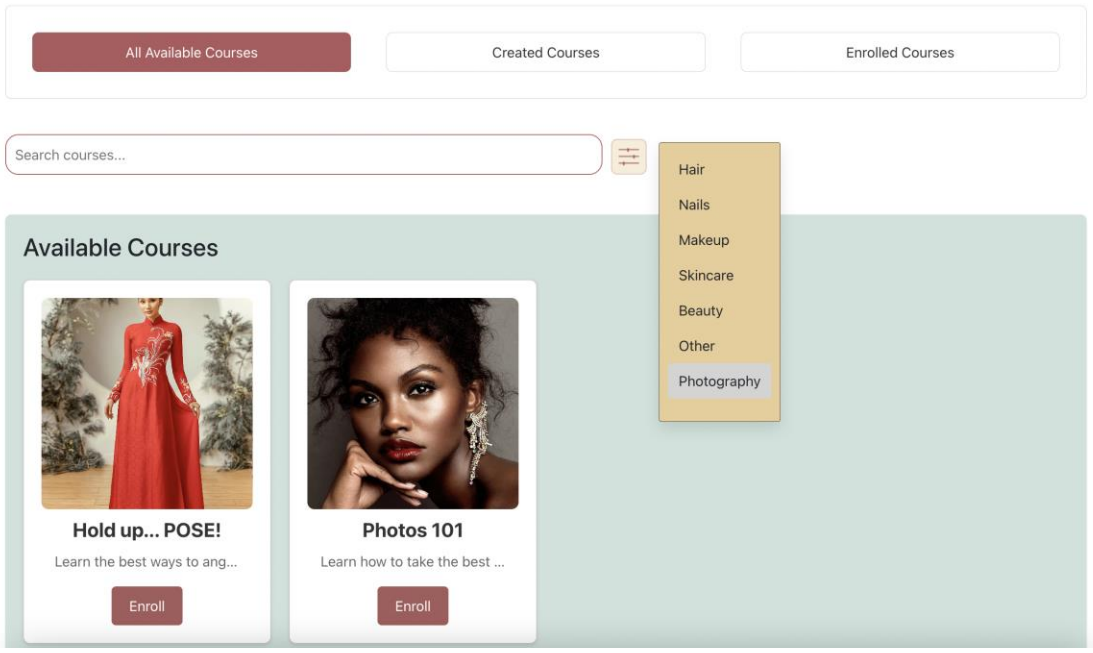
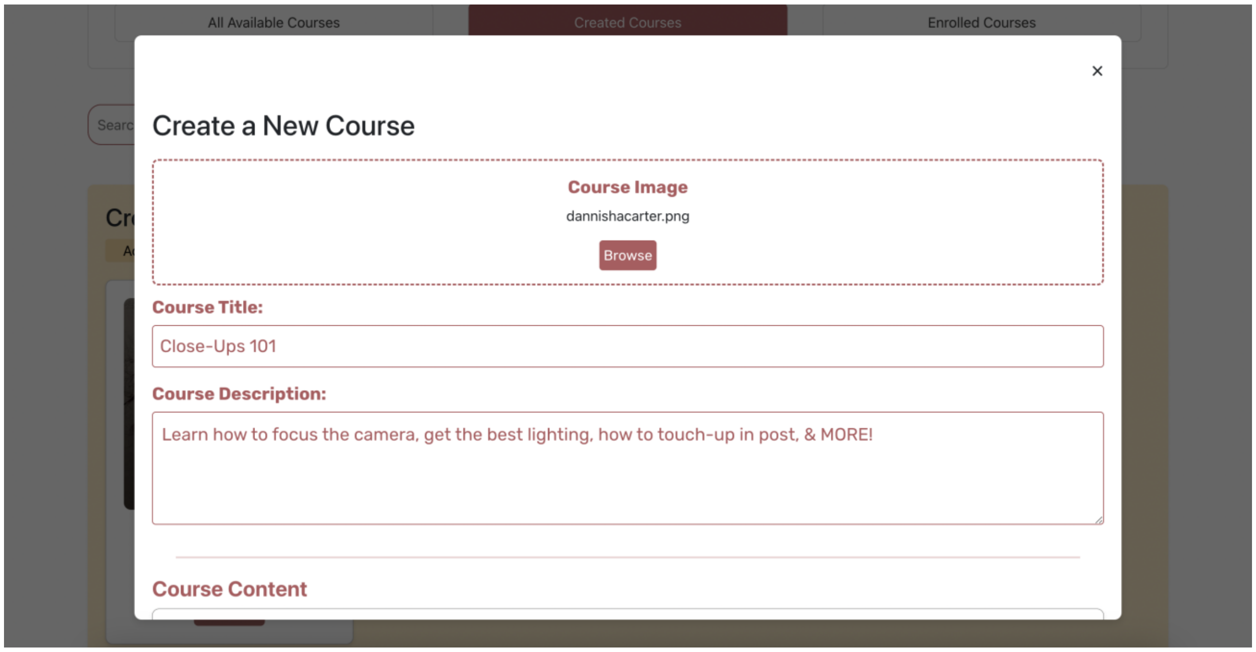


Figure 13: Available Courses

Users with the course creator role also have the ability to add new courses through a pop-up screen that prompts them to enter relevant course information.



Create a New Course

Course Image
dannishacarter.png

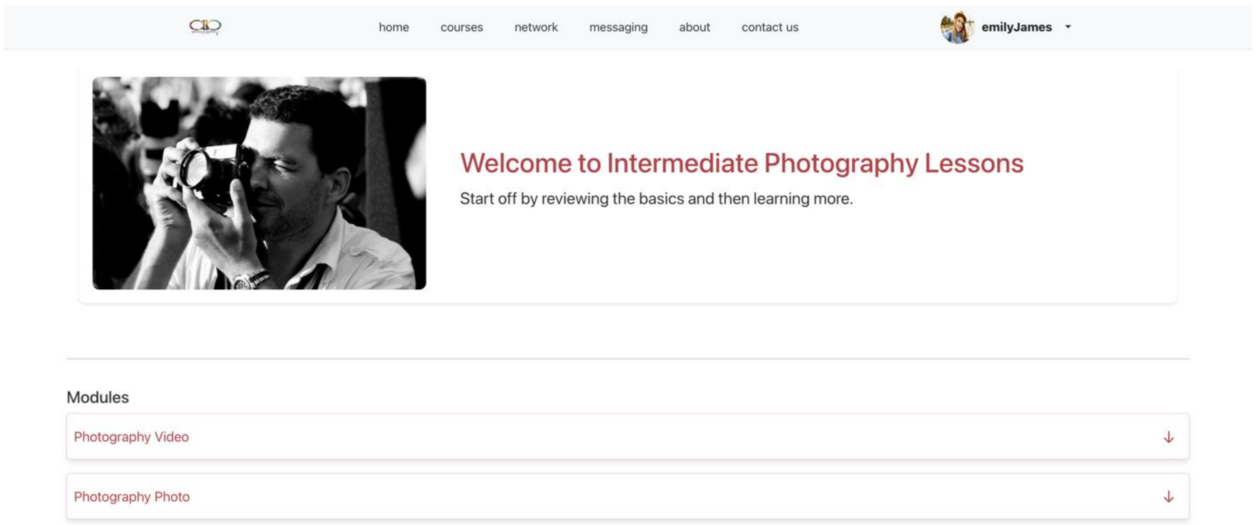
Course Title: Close-Ups 101

Course Description: Learn how to focus the camera, get the best lighting, how to touch-up in post, & MORE!

Course Content

Figure 14: Create Course

Once a course is made, users can click to view the content and choose to enroll in the course if they are interested in learning more.



Welcome to Intermediate Photography Lessons

Start off by reviewing the basics and then learning more.

Modules

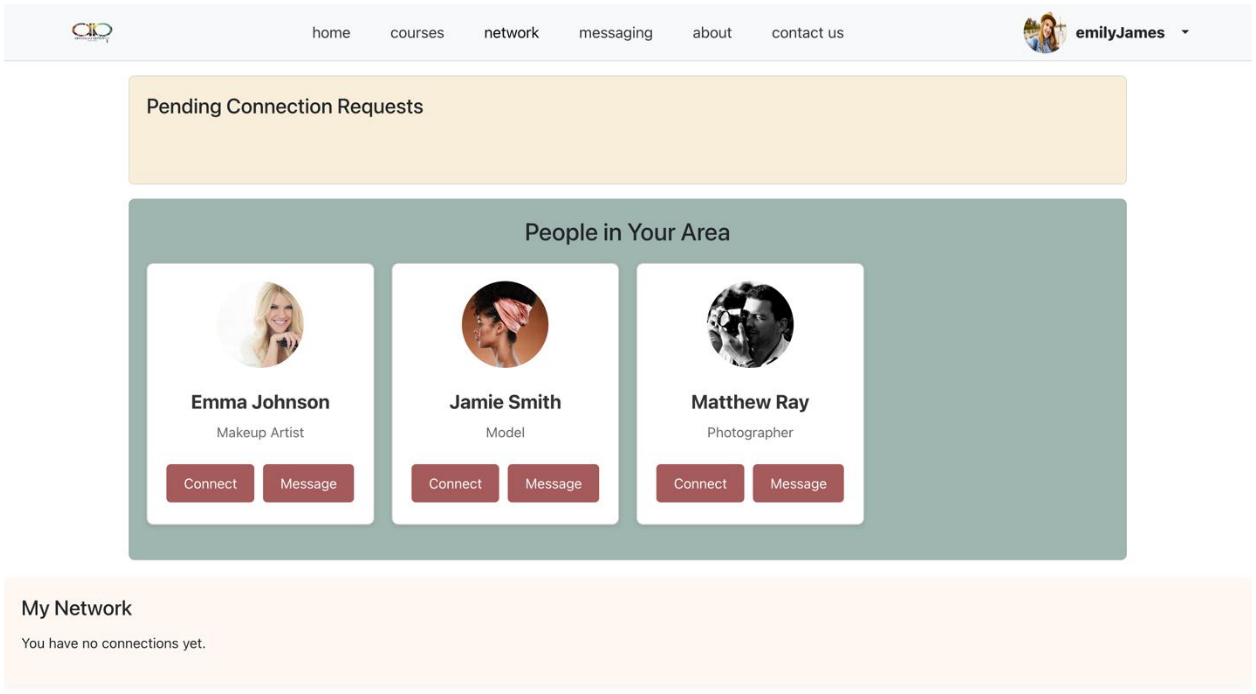
Photography Video

Photography Photo

Figure 15: Course Page

Network

A very important aspect of our website are the networking and community features. On the Network page, users can see a list of other users in their area.



Pending Connection Requests

People in Your Area

Emma Johnson
Makeup Artist

Jamie Smith
Model

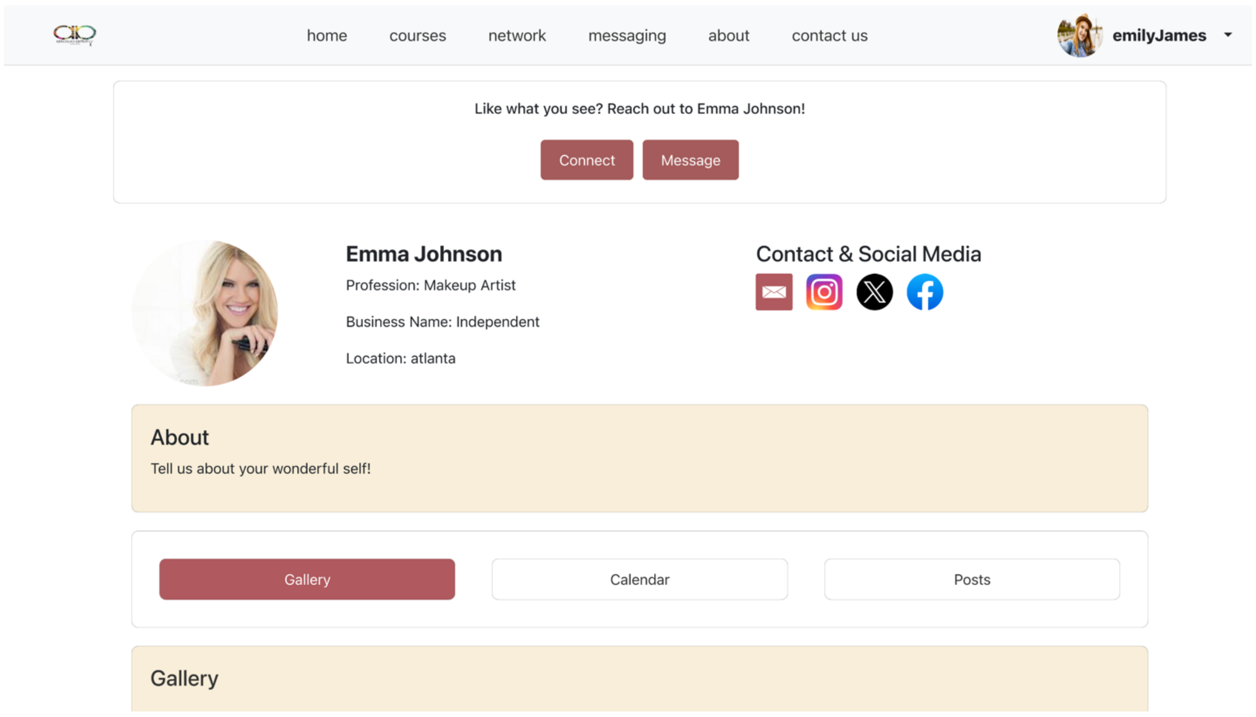
Matthew Ray
Photographer

My Network

You have no connections yet.

Figure 16: Network

To learn more about a certain person, users can click on their name to view their profile. If a user wants to form a connection with someone, they can click the connect button to send a request.



Like what you see? Reach out to Emma Johnson!

Emma Johnson

Profession: Makeup Artist

Business Name: Independent

Location: atlanta

Contact & Social Media

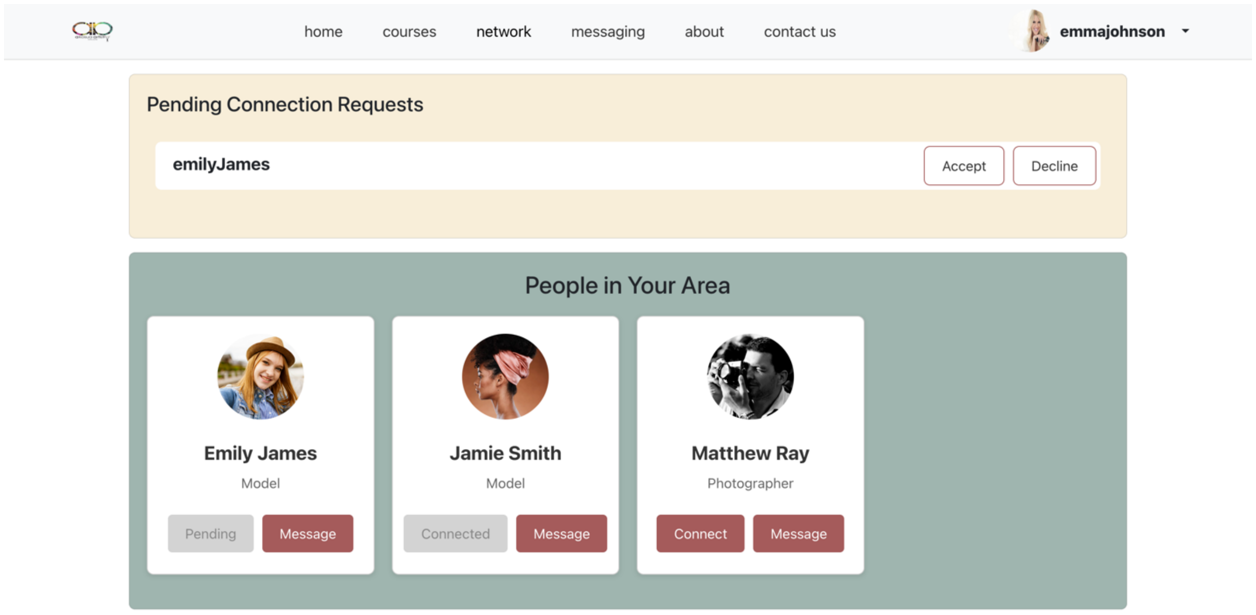
About

Tell us about your wonderful self!

Gallery

Figure 17: Other User Profile

Users can view a list of their pending requests and choose whether to accept or decline the connection.



Pending Connection Requests

emilyJames

People in Your Area

Emily James
Model

Jamie Smith
Model

Matthew Ray
Photographer

Figure 18: Connection Requests

Messaging

If a user wants to send a message, they will be able to click the message button on the Network or the other user's Profile Page and be redirected to the Messaging page.

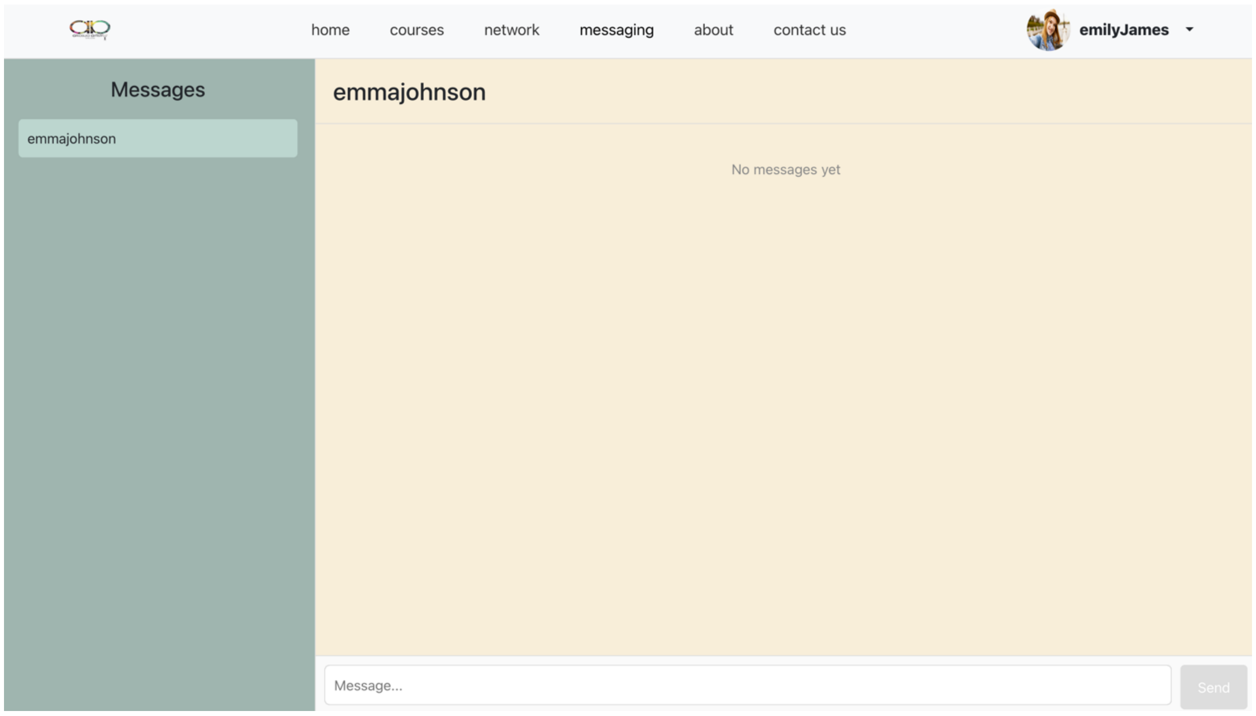
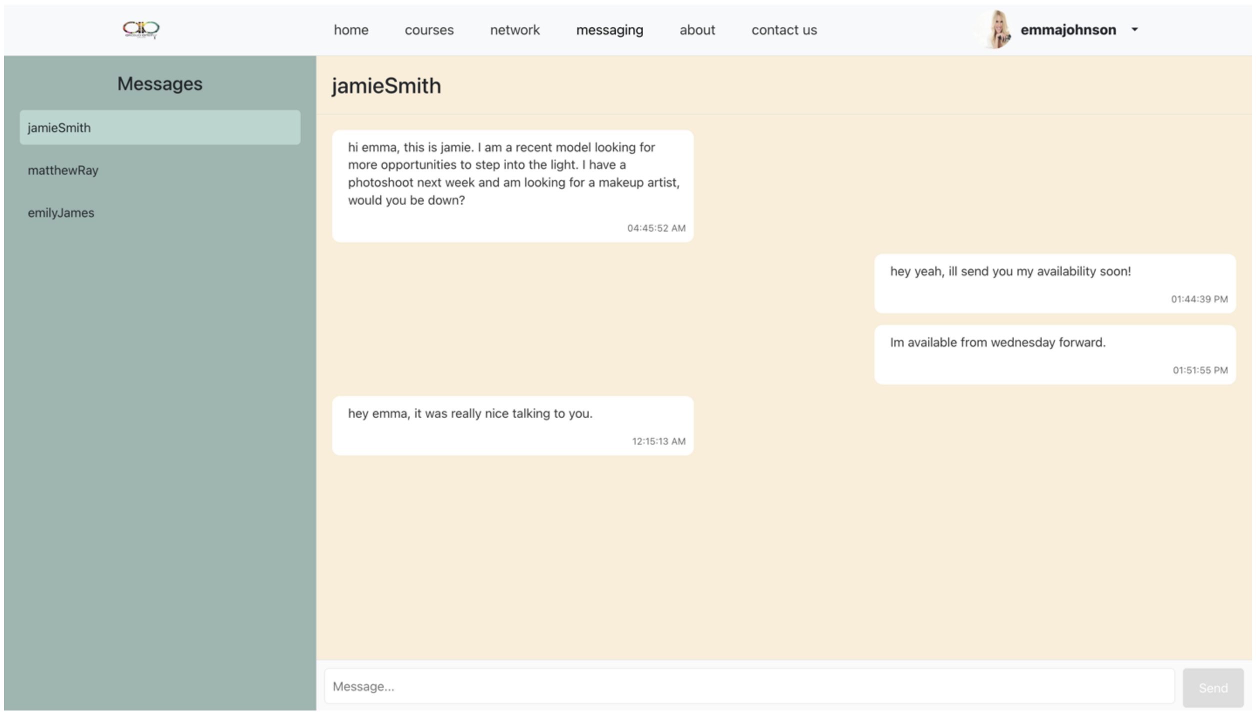


Figure 19: Messaging

Once a chat is open with another user, messages can be sent back and forth. This way, users can communicate directly with people they want to learn more about. Users can view all the chats they are part of on the sidebar, and click on a different name to toggle to that conversation.



jamieSmith

hi emma, this is jamie. I am a recent model looking for more opportunities to step into the light. I have a photoshoot next week and am looking for a makeup artist, would you be down?
04:45:52 AM

hey yeah, ill send you my availability soon!
01:44:39 PM

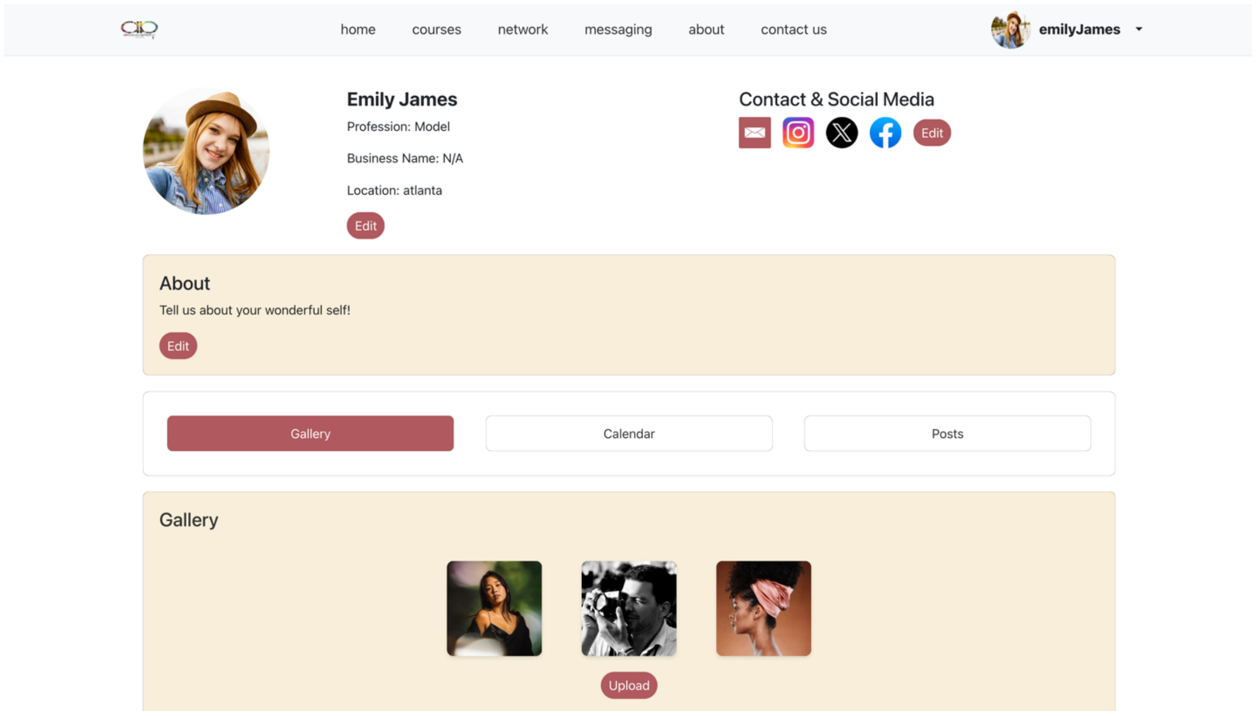
Im available from wednesday forward.
01:51:55 PM

hey emma, it was really nice talking to you.
12:15:13 AM

Figure 20: Ongoing Conversation

Profile

In the profile page, users can manage how they want to represent themselves to other users on the webpage. This includes allowing them to provide their name, profession, social media, and location. Additionally, they can provide a short "About" description of themselves.



Emily James
Profession: Model
Business Name: N/A
Location: atlanta

Contact & Social Media

About
Tell us about your wonderful self!

Gallery

Figure 21: Profile

Gallery

When toggled to "Gallery", users can upload photos to their personal gallery, which is viewable by others.

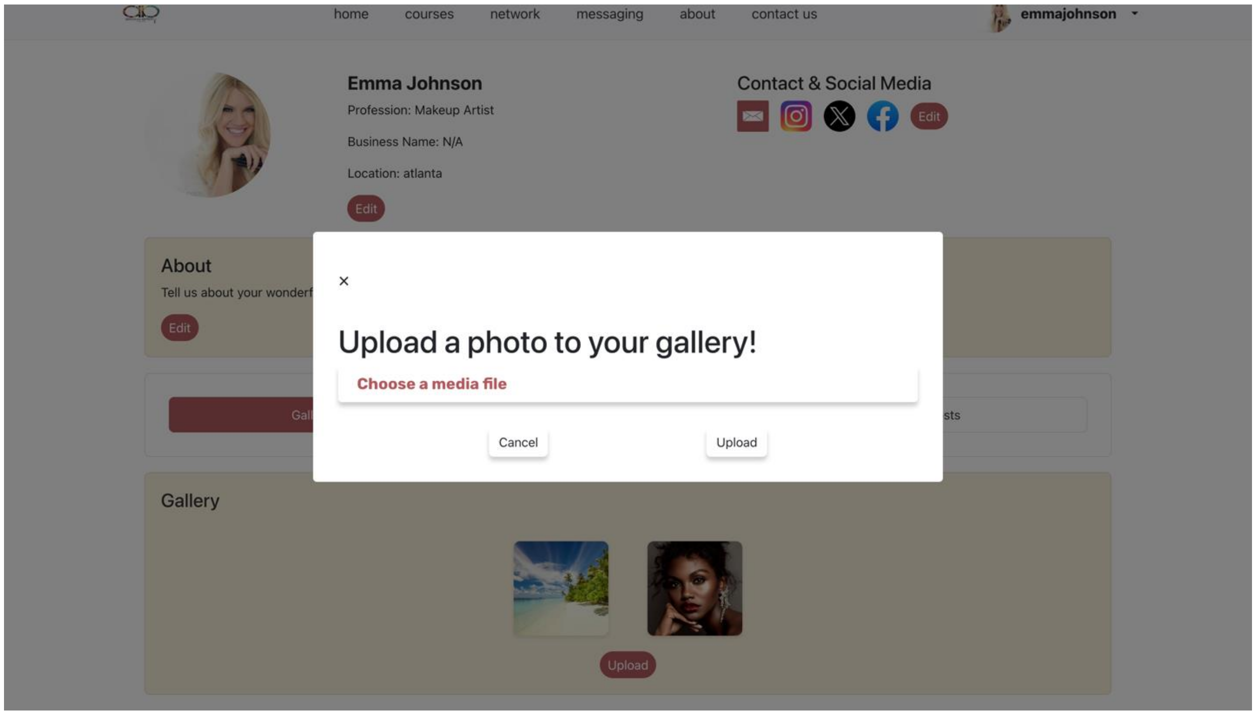
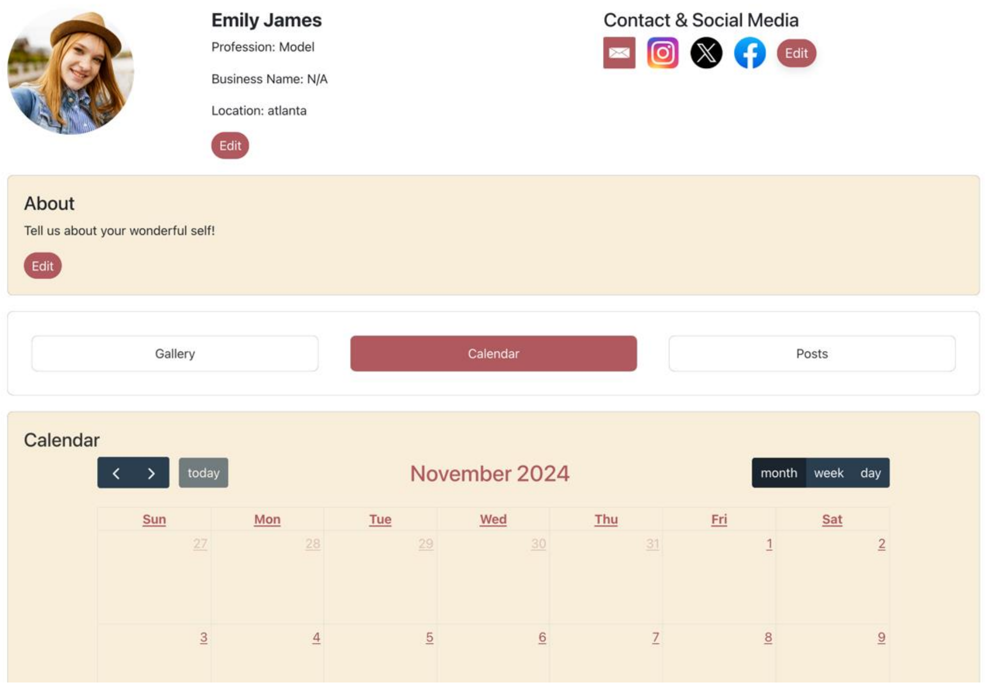


Figure 22: Gallery Upload

Calendar

After toggling to “Calendar”, users can input events into their personal calendar. Others will be able to view these events when clicking on profile.



Emily James

Profession: Model

Business Name: N/A

Location: atlanta

Contact & Social Media

About

Tell us about your wonderful self!

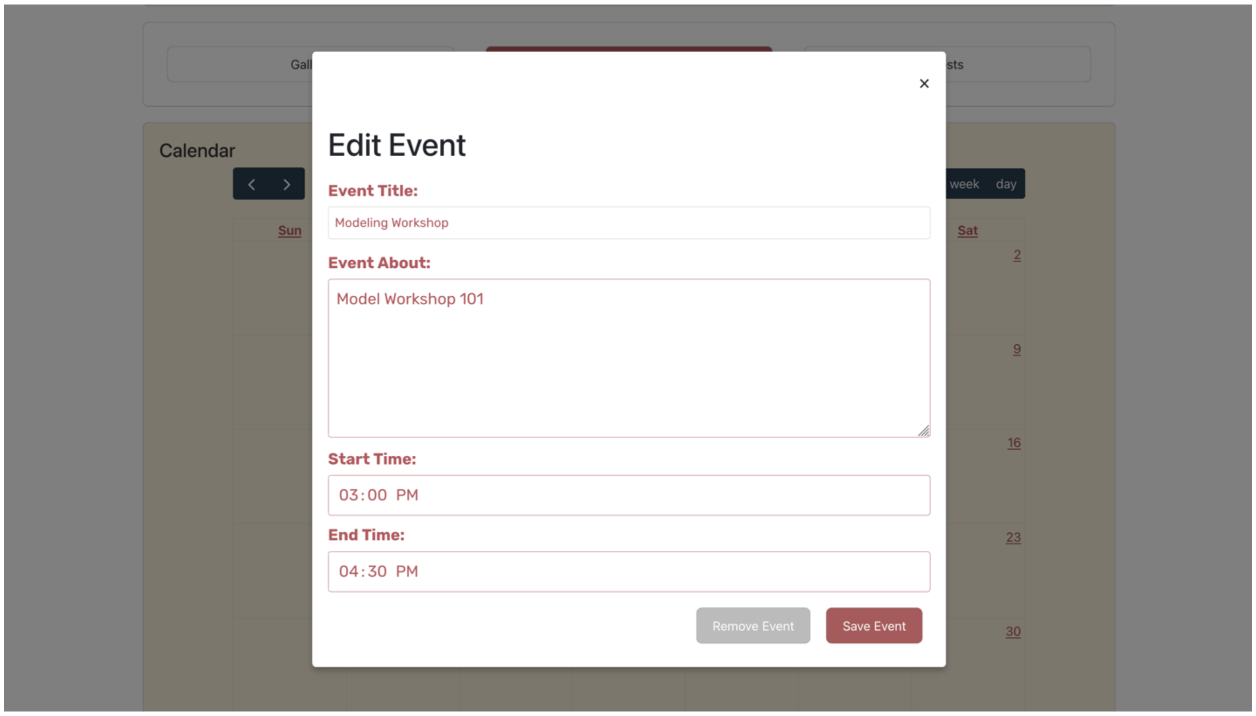
Calendar

November 2024

Sun	Mon	Tue	Wed	Thu	Fri	Sat
27	28	29	30	31	1	2
3	4	5	6	7	8	9

Figure 23: Calendar

When creating an event, users can add a title, description, and event times.


Edit Event

Event Title:

Modeling Workshop

Event About:

Model Workshop 101

Start Time:

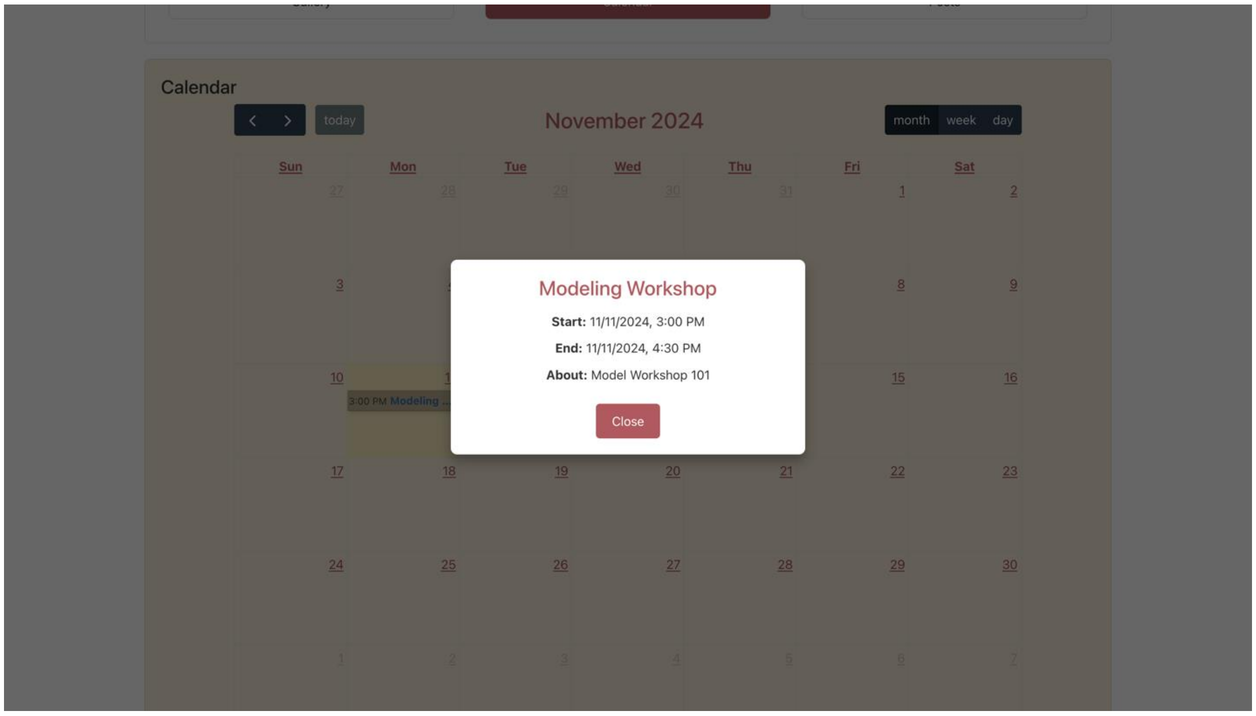
03:00 PM

End Time:

04:30 PM

Figure 24: Add Event

Once created, events can be clicked to view the description and other details.


Modeling Workshop

Start: 11/11/2024, 3:00 PM

End: 11/11/2024, 4:30 PM

About: Model Workshop 101

Figure 25: View Event Details

Posts

Under “Posts”, there is a display of every post made by the user whose profile is being viewed.

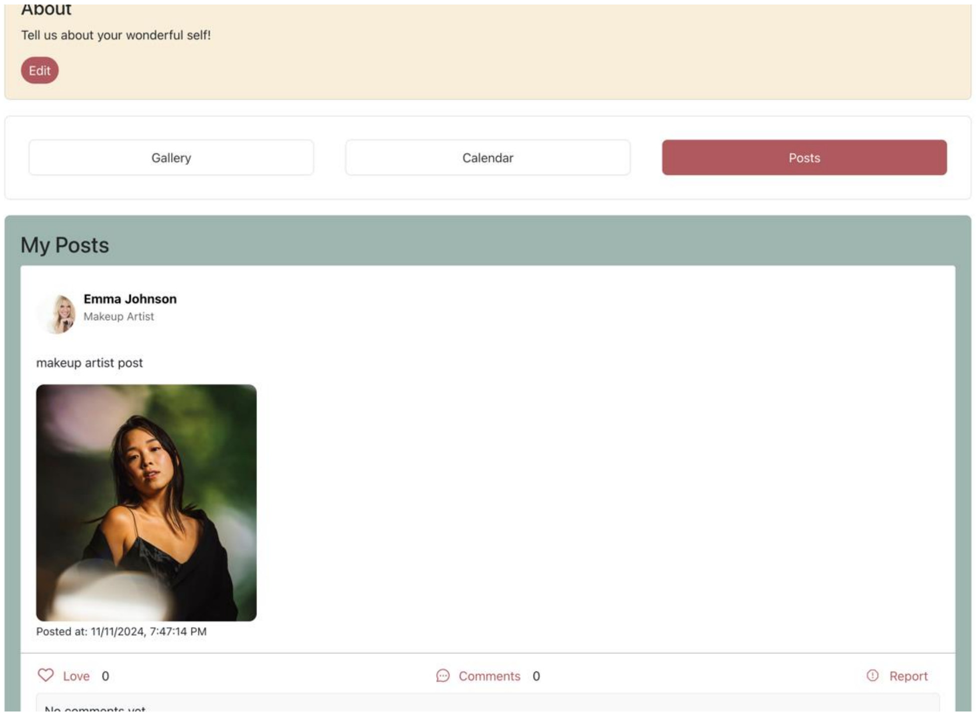


Figure 26: User Posts

Appendix

Axios

Library URL: https://axios-http.com/docs/intro

Library Purpose: Axios helped us handle any HTTP request between our React and Flask where we used it to fetch/post user data such as posts, connections, courses, and events

Example:

```
Axios.post(‘/api/create_event’, {
”user_id:” “user123”,
“title”: “Meeting ”,
“about”: “DD Design Meeting”,
“year”: 2024,
“month”: 11,
“day”: 17,
“start_time”: “12:00”,
“end_time”: “17:00”
}
```

Body-Lock-Scroll

Library URL: https://github.com/willmcpo/body-scroll-lock

Library Purpose: This library was used to help us manage the scrolling behavior on our website when certain modals were open to help prevent background context from scrolling, improving user experience.

Example:

```
import { disableBodyScroll, enableBodyScroll } from 'body-scroll-lock';
const Modal = ({ isOpen, onClose }) => {
   const modalRef = useRef();
   useEffect(() => {
      if (isOpen) {
         disableBodyScroll(modalRef.current);
      } else {
         enableBodyScroll(modalRef.current);
      }
   }, [isOpen]);
```

Date-FNS

Library URL: https://date-fns.org/

Library Purpose: Date-FNS is a library that helped us format our date when user had selected the date/time for their events in a way where we can digest it and use it throughout our event editing/displaying methods.

Example:
```
import { format } from ‘date-fns’;
const eventDate = new Date(2024, 10, 17);
const formattedDate = format(eventDate, 'yyyy-MM-dd');
```

React-Bootstrap

Library URL: https://react-bootstrap.github.io/

Library Purpose: This library helps give pre-styled UI components for React applications, where it helps us build visually appealing components like dropdowns, modals, and navigation bars.

Example:
```
import { Dropdown } from 'react-bootstrap';

<Dropdown>
<Dropdown.Toggle> Choose </Dropdown.Toggle>
<Dropdown.Menu>
   <Dropdown.Item> Action </Dropdown.Item>
   </Dropdown.Menu>
</Dropdown>;
```

FullCalendar

Library URL: https://fullcalendar.io/docs/react

Library Purpose: This library is used for implementing our interactive calendar where users can view, add, edit, and manage events easily on the calendar interface.

Example:

```
import FullCalendar from '@fullcalendar/react';
Import dayGridPlugin from '@fullcalendar/daygrid';
import interactionPlugin from '@fullcalendar/interaction';

const MyCalendar = () => {
   const events = [
      { id: 1, title: 'Meeting', start: '2024-11-17T10:00:00', end: '2024-11-17T11:00:00' },
      { id: 2, title: Expo, start: '2024-12-02T09:00:00', end: '2024-11-02T16:00:00' }
];
   const handleDateClick = (info) => {
      console.log('Selected Date:', info.date);
```

```
  };
  const handleEventClick = (info) => {
    console.log('Event Details:', info.event);
  };
  return (
    <FullCalendar
      plugins={[dayGridPlugin, interactionPlugin]}
      initialView="dayGridMonth"
      events={events}
      dateClick={handleDateClick}
      eventClick={handleEventClick}
    />
  );
};
```

React-Responsive-Carousel

Library URL: https://www.npmjs.com/package/react-responsive-carousel

Library Purpose: This library is used to create interactive and responsive carousels/slideshows for displaying images/content on our website.

Example:

```
import { Carousel } from 'react-responsive-carousel';

<Carousel>
  <div><img src="hairTutorial.jpg" alt="Slide 1" /></div>
  <div><img src="stylingTutorial.jpg" alt="Slide 2" /></div>
</Carousel>;
```

React-Router-Bootstrap

Library URL: https://react-router-bootstrap.github.io/

Library Purpose: This library helps use certain Bootstrap components like NavLink, making navigation easy.

Example:

```
import { LinkContainer } from 'react-router-bootstrap';
<LinkContainer to="/home">
  <Nav.Link>Home</Nav.Link>
</LinkContainer>;
```

React-Router-Bootstrap

Library URL: https://reactrouter.com/

Library Purpose: This library helps us handle the different routing within our web application, enabling navigation between our different pages/JS files.

Example:

```
import { BrowserRouter as Router, Route, Routes, Link } from 'react-router-dom';

<Router>
   <nav>
      <Link to="/home">Home</Link>
      <Link to="/about">About</Link>
   </nav>
   <Routes>
      <Route path="/home" element={<Home />} />
      <Route path="/about" element={<About />} />
   </Routes>
</Router>;
```

Socket.IO-Client

Library URL: https://socket.io/

Library Purpose: This library was used to help us implement our real time, bi-directional messaging feature where users can communicate with one another between the client/server.

Example:

```
import { io } from 'socket.io-client';

const socket = io('http://localhost:5000');
socket.on('connect', () => {
   console.log('Connected to server');
});
socket.on('message', (data) => {
   console.log('Message received:', data);
});
socket.emit('sendMessage', { text: 'Hello, server!' });
```