Ambrose Treacy College Blogging Website

27/08/2023
NATHAN PERRIER
AMBROSE TREACY COLLEGE

Table of Contents

Explore

Problem Description

- Ambrose Treacy College is an all-boys school located in the Brisbane area who has requested a website.
- The Senior English department requires a writing platform for their blogging project. You are required to build a prototype blogging website application.
- This application should allow a user to log in and create a blog post.
- These blogs posts should then be viewed by other site members and the teacher.
- The website should also allow the login of an administrator (teacher) with extended privileges.

Development Platform

- To develop the final products multiple languages could be used.
- The languages chosen for this project are; Python (backend), JavaScript (frontend + backend), PHP (backend), SQLite (backend (database)), HTML (frontend), CSS (frontend), SCCS (frontend), Bootstrap CSS (frontend)
- Some of the libraries used in this project are; SQLAlchemy, Flask, Bcrypt, OpenAI, Flask Login, PIL, Google Maps, and more.
- The amalgamation of these languages and libraries should produce the final product.

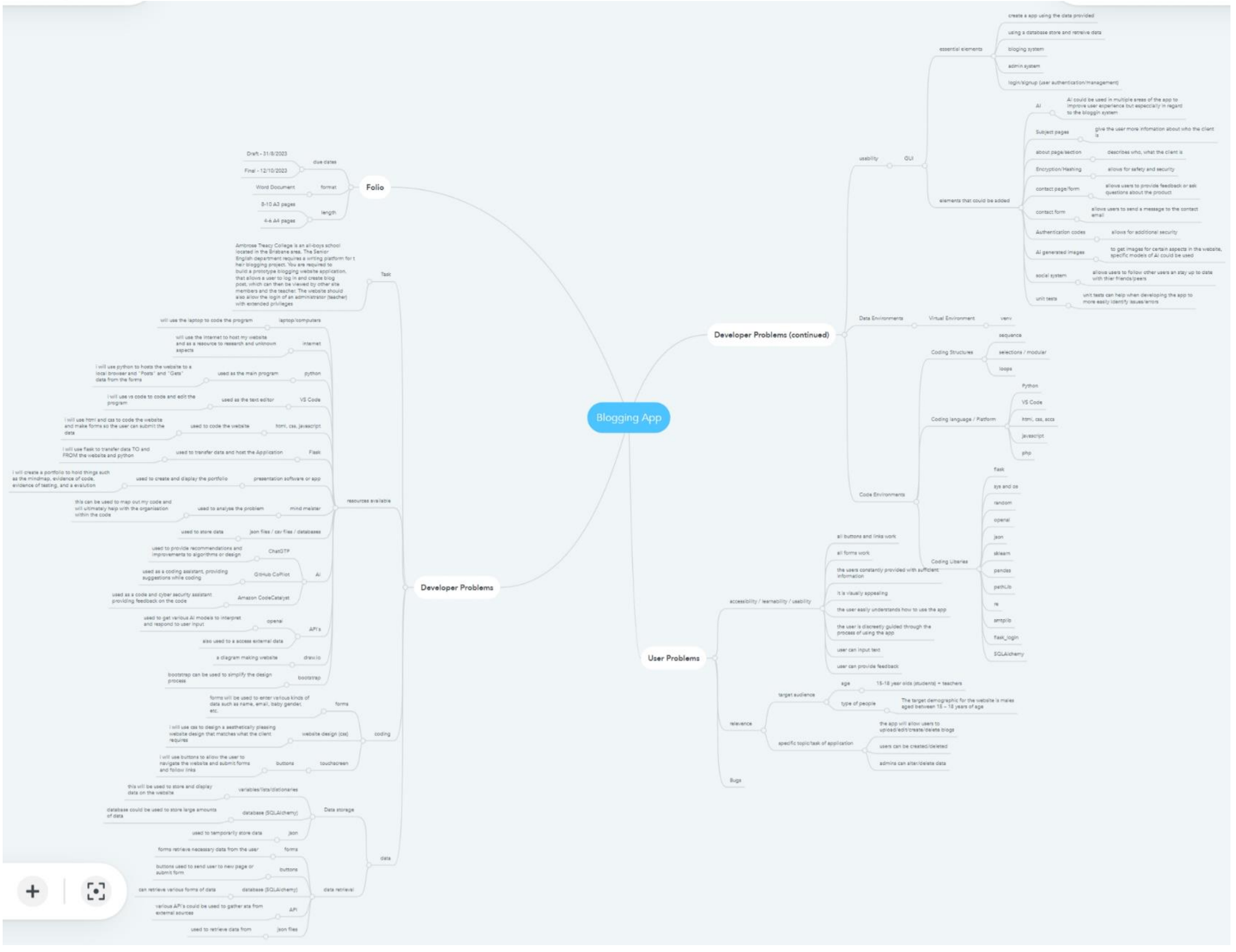
Project Scope

- Deadlines:
  - Draft Deadline: Week 8, Term 3
  - Final Deadline: Week 2, Term 4
- The Device used is adequate and has the necessary libraries and languages installed
- The website will be developed for the popular web browsers

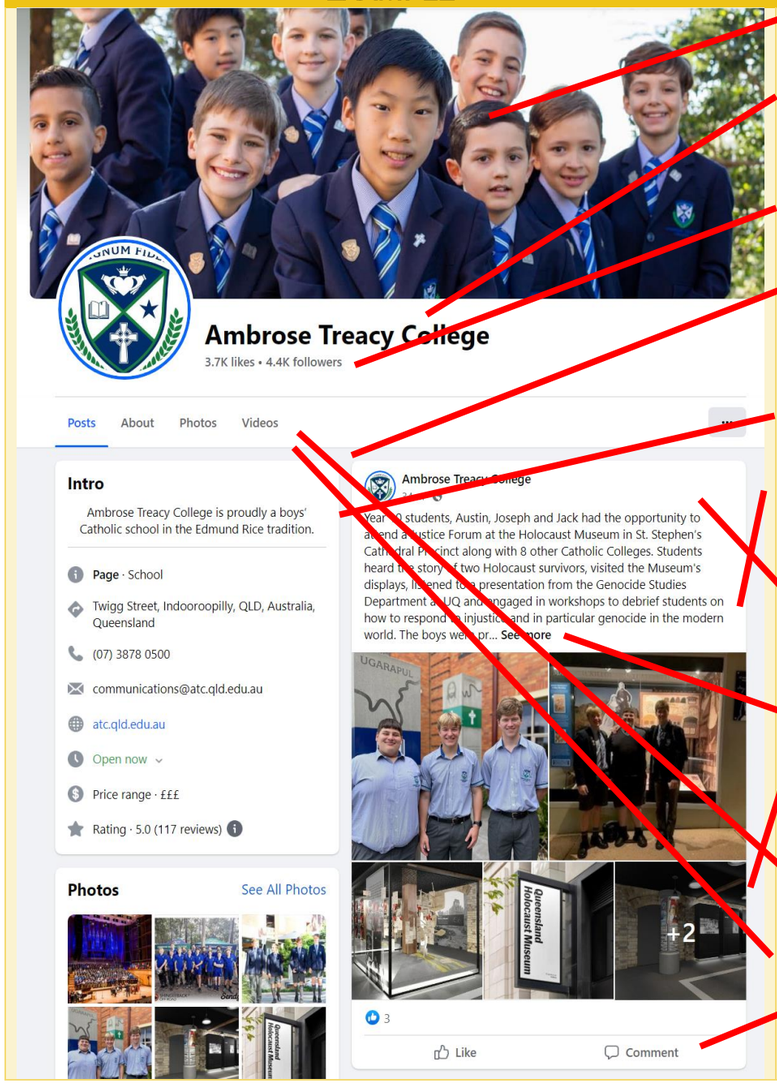
Development Platform

PRESCRIBED CRITERIA		SELF-DETERMINED CRITERIA	
PC1	Develop a website with many pages and features	SDC1	Using objects to code in a modular way
PC2	Allow multiple users to use the website and have the ability for new users to be added	SDC2	Using lists, arrays, and dictionaries to store data
PC3	Use databases for data storage	SDC3	Allow users to upload and customise their profile picture
PC4	Create a folio that outlines the user interface and coded components of the solution	SDC4	Use authentication codes to confirm user authenticity
PC5	Have blog functionality that all users, can use, view and access	SDC5	Send emails to users to send necessary information
PC6	Generate a user/visual interface	SDC6	Use JS for data handling and design elements
PC7	Incorporate the use of the client's signature colours or logo	SDC7	Allow the users to reset their passwords if forgotten
PC8	Integrate security features such as hashing and encryption	SDC8	Develop and use the Lowenstein distance algorithm
PC9	Users must be able to edit or delete their information	SDC9	Integrate a spell check in the blog functionality
PC10	Have admin functionality that is given to the client's staff. They should be able to edit, delete and add user and blog data.	SDC10	Have nested comments for blog that users can edit and delete
PC11	Allow for users to login, signup and logout	SDC11	Use SQLAlchemy and a multi-file architecture for the code design
		SDC12	Use CSS to style the website to adhere closely with the client's style
		SDC13	Use animations on the website to engage the user
		SDC14	Prioritise website responsiveness to improve accessibility
		SDC15	Have a like-dislike system for blogs and comments
		SDC16	Use AI to generate blog images if the user has no image to upload
		SDC17	Have a social system where users can follow/unfollow other users
		SDC18	Develop a notification system where user's will be notified when they are followed or a user they are following uploads a blog
		SDC19	Have an AI to suggest words or sentences as the user is typing
		SDC20	Have an AI to generate blog titles or descriptions
		SDC21	Allow for user's to add tags or categories to their blogs to allow to inform readers and add context. This will also be used for the search feature
		SDC22	Allow users to search for blogs or users
		SDC23	Have a featured blog section were the blogs with the most likes or engagement (number of comments) will appear
		SDC24	Allow users to achieve their blog posts or save as a draft when creating one
		SDC25	Use drone footage of the Client's campus to engage users when they click on the website
		SDC26	Have terms and policies regarding the use of the website. Make users agree to these terms before signing up
		SDC27	Have multiple pages regarding information about the client such as subjects or their relation to EREA
		SDC28	Have a contact form where users can ask questions
		SDC29	Have a newsletter where users can sign up and receive newsletters
		SDC30	Have unit tests for various programming elements
		SDC31	Have activity logs that the user and admin can view
		SDC32	Have tracking functionality that identifies the user's or a device's likes in terms of the blogs they view. Use this data to recommend blogs to the user.
		SDC33	Have AI generated and default profile pictures gathered from API's.
		SDC34	Allow admins to search for users based on their id, username, name or email

Mind Map

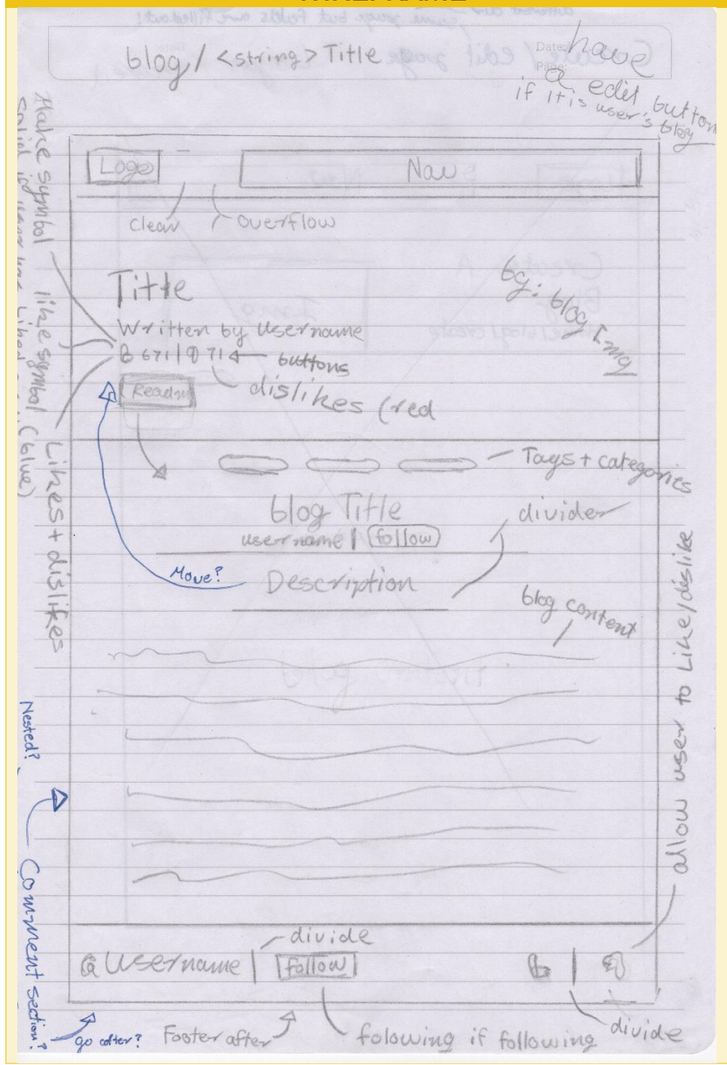

Blogging App
Folio
due dates
Draft - 31/8/2023
Final - 12/10/2023
format
Word Document
length
8-10 A3 pages
4-6 A4 pages
Task
Ambrose Treacy College is an all-boys school located in the Brisbane area. The Senior English department requires a writing platform for t heir blogging project. You are required to build a prototype blogging website application, that allows a user to log in and create blog post, which can then be viewed by other ste members and the teacher. The website should also allow the login of an administrator (teacher) with extended privileges
Developer Problems
resources available
laptop/computers
will use the laptop to code the program
internet
will use the internet to host my website and as a resource to research and unknown aspects
python
used as the main program
I will use python to hosts the website to a local browser and "Posts" and "Gets" data from the forms
VS Code
used as the text editor
I will use vs code to code and edit the program
html, css, javascript
used to code the website
I will use html and css to code the website and make forms so the user can submit the data
Flask
used to transfer data and host the Application
I will use flask to transfer data TO and FROM the website and python
presentation software or app
used to create and display the portfolio
I will create a portfolio to hold things such as the mindmap, evidence of code, evidence of testing, and a evalution
mind meister
used to analyse the problem
this can be used to map out my code and will ultimately help with the organisation within the code
json files / csv files / databases
used to store data
AI
ChatGTP
used to provide recommendations and improvements to algorithms or design
GitHub CoPilot
used as a coding assistant, providing suggestions while coding
Amazon CodeCatalyst
used as a code and cyber security assistant providing feedback on the code
API's
openai
used to get various AI models to interpret and respond to user input
also used to a access external data
draw.io
a diagram making website
bootstrap
bootstrap can be used to simplify the design process
coding
forms
forms will be used to enter various kinds of data such as name, email, baby gender, etc.
website design (css)
I will use css to design a aesthetically pleasing website design that matches what the client requires
touchscreen
buttons
I will use buttons to allow the user to navigate the website and submit forms and follow links
data
Data storage
variables/lists/dictionaries
this will be used to store and display data on the website
database (SQLAlchemy)
database could be used to store large amounts of data
json
used to temporarily store data
data retrieval
forms
forms retrieve necessary data from the user
buttons
buttons used to send user to new page or submit form
database (SQLAlchemy)
can retrieve various forms of data
API
various API's could be used to gather ata from external sources
json files
used to retrieve data from
Developer Problems (continued)
usability
GUI
essential elements
create a app using the data provided
using a database store and retreive data
blogging system
admin system
login/signup (user authentication/management)
elements that could be added
AI
AI could be used in multiple areas of the app to improve user experience but especially in regard to the bloggin system
Subject pages
give the user more information about who the client is
about page/section
describes who, what the client is
Encryption/Hashing
allows for safety and security
contact page/form
allows users to provide feedback or ask questions about the product
contact form
allows users to send a message to the contact email
Authentication codes
allows for additional security
AI generated images
to get images for certain aspects in the website, specific models of AI could be used
social system
allows users to follow other users an stay up to date with thier friends/peers
unit tests
unit tests can help when developing the app to more easily identify issues/errors
Data Environments
Virtual Environment
venv
Code Environments
Coding Structures
sequance
selections / modular
loops
Coding language / Platform
Python
VS Code
html, css, sccs
javascript
php
Coding Libaries
flask
sys and os
random
openai
json
sklearn
pandas
pathLib
re
smtplib
flask_login
SQLAlchemy
User Problems
accessibility / learnability / usability
all buttons and links work
all forms work
the users constantly provided with sufficient information
it is visually appealing
the user easily understands how to use the app
the user is discreetly guided through the process of using the app
user can input text
user can provide feedback
relevance
target audience
age
15-18 year olds (students) + teachers
type of people
The target demographic for the website is males aged between 15 – 18 years of age
specific topic/task of application
the app will allow users to upload/edit/create/delete blogs
users can be created/deleted
admins can alter/delete data
Bugs

Existing Solutions

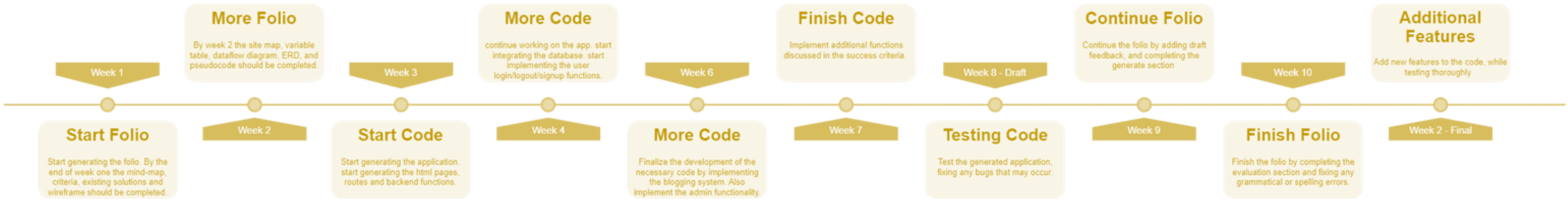
EXAMPLE	ANNOTATIONS
	Has a cover image of the user, allowing them to provide addition visual feedback. Displays the name of the user is large font to improve the accessibly of the page. The profile image of the user is displayed at the top. It is sat in the middle of the two divs. The amount of likes and followers that the user has is displayed under their name. The layout of this page is neat, with all navigation being easy to find and all sections being placed logically. This improves the learnability of the page. An information card about the user is displayed to inform visitors of who they are, where to find them and how to contact them. All the user's posts are neatly organised in a column, with images being displayed under the text. The post do not allow for titles, only body text The posts allow for more than one image to be displayed. This allows for more information to be provided. Posts have a 'see more' button if there is a lot of text. This makes it so posts do not take up too much space. The webpage is responsive and can be displayed on all screen sizes. This improves the utility and accessibility of the app (SDC14). The webpage has multiple sections about the user, such as, 'about', 'posts', 'photos' and 'videos'. This makes it so everything is not cramped into one page. The page allows for videos to be uploaded. Users can like or comment on posts (SDC15)(SDC10). Comments are nested so that visitors can reply to other comments (SDC10).

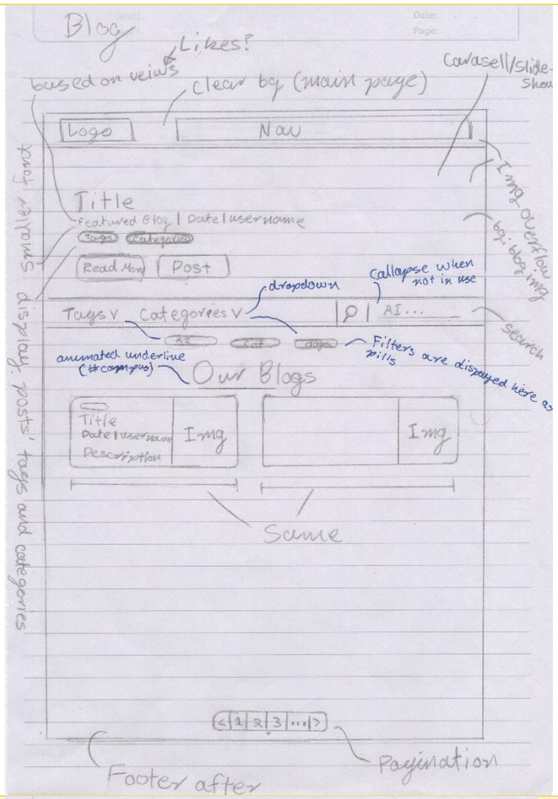
Develop

Wireframes

WIREFRAME	ANNOTATIONS
	The navbar is always displayed at the top of the page to improve the learnability of the site. The image of the blog will be displayed as the background at the top of the page and will overflow under the navbar. The title, description, author, and like/dislike count will be displayed at the top of the page. There is a 'read more' button to improve the utility of the page. The tags and categories of the blog will be displayed about the title (SDC21). The title, author and description will be displayed again above the blog content. A follow button will be displayed next to the authors username (SDC17). The usernames will be links to their account page. The layout of the site is neat and modern to improve the utility and learnability of the site (SDC12). Text will use appropriate contrast to improve accessibility. A social section will be displayed under the content where the authors username will be displayed once more. Users can also follow/unfollow the author here as well as like/dislike the blog (SDC17) (SDC15). A comments section will be displayed under the social section. A user must be logged in to comment, reply, edit comment, like. Dislike and follow/unfollow. They will be redirected to login if they try to use these features. If the author views their own blog, an 'edit blog' button will be displayed instead of a follow button (PC10). Comments will be nested so users can reply to other comments (SDC10). Users can edit or reply to their own comments (SDC10). User's can also like/dislike comments

Project Timeline





Featured blogs will be displayed at the top of the page (SDC23). These blogs are gathered from the amount of likes – dislikes. The blogs title, author, date, and categories/tags are displayed. Users can read more by pressing the 'read more' button.

The featured blogs are displayed on a slideshow animation to maximise the number of blogs that can be displayed, and to engage the user (SDC13).

Blogs can be filtered using the filter section where blogs can be searched (SDC22), or filtered by their tags or categories.

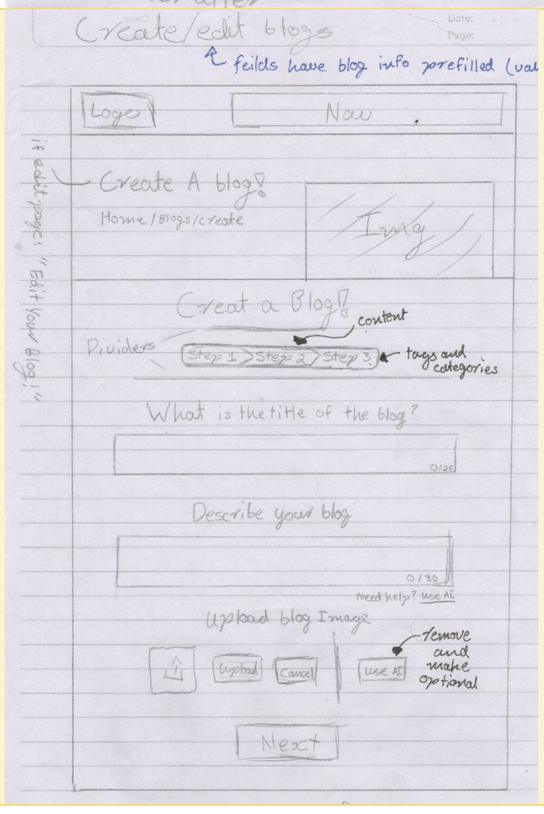
Active filters applied will be displayed at the top of the blogs section, to inform the user about the active filters. This is to improve learnability.

The responsiveness of this page will be adjusted to all screen sizes to improve accessibility (SDC14)

Each blog will be displayed on a card, where their most relevant category of tag (gathered using AI), will be displayed at the top. Below that is the blog title. The blogs author and date will be displayed on the card. Finally, the blogs description will also be displayed.

To ensure the page does not go on forever, pagination will be used to ensure only 6 blogs are displayed at a time.

Users can navigate the pages by using the page navbar at the bottom.



The title of the page is displayed at the top in a large font to improve learnability and accessibility.

Both the edit and create page use the same layout. However, the edit page has the blogs values inserted into the input fields.

This form uses multiple fields to minimise the amount of vertical space needed. Is allow allows for validation to be done at each step, using JavaScript (SDC6).

The steps are displayed at the top and will indicate what step the user is currently on. This was to improve learnability.

Users can go to previous steps if needed to improve utility.

The form inputs are displayed with labels to improve learnability.

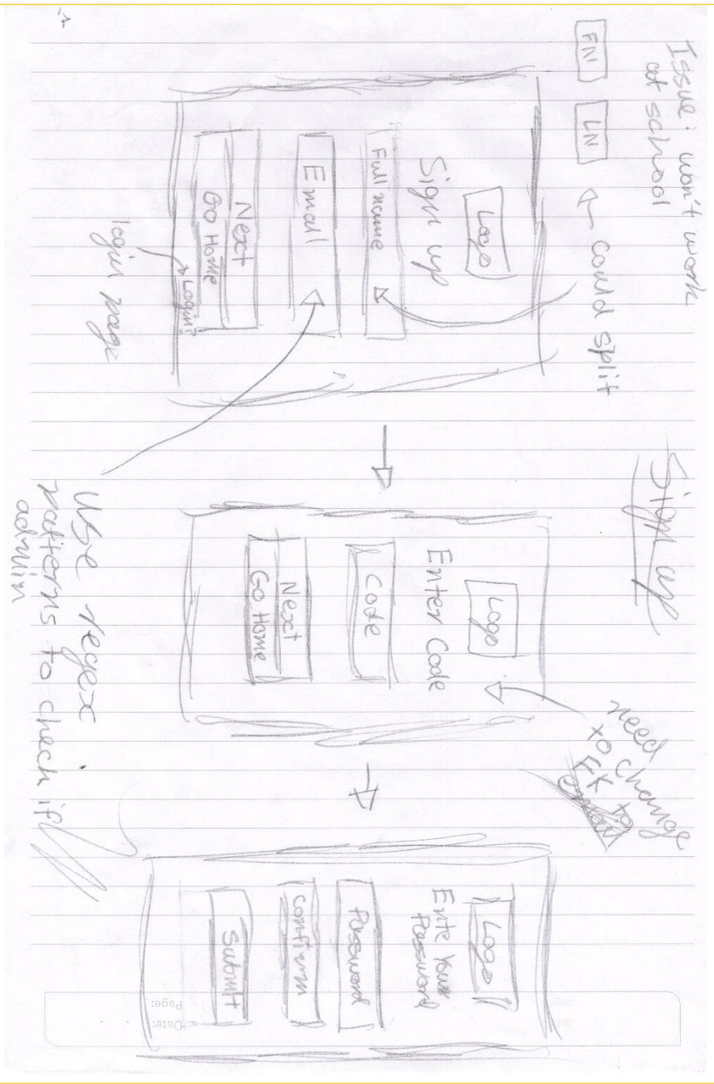
AI is used to provide suggestions, based on their current input, as the user types (SDC19).

An AI spell check is used once the form is submitted to fix and mistakes (SDC9).

Users can upload their own images or use AI images (SDC16).

These images are created based on the description of the blog.

Users can also save as drafts or achieve the blogs (SDC24)



The client's logo is displayed on each step of the signup form to improve learnability.

The signup form uses three steps to validate each section before continuing. Furthermore, it allows for the email from the first section to be used to send the code for the second section (SDC5).

The head of each section informs the user of what to do to improve learnability.

The form is responsive and can fit any screen size to improve accessibility.

The user can go back if they put in the wrong information.

Error messages are clearly displayed and use a appropriate contrast to improve accessibility.

The steps of the form will be displayed at the bottom to inform the user of what step they are on. This is to improve learn ability.

Users must accept the terms and conditions before signing up.

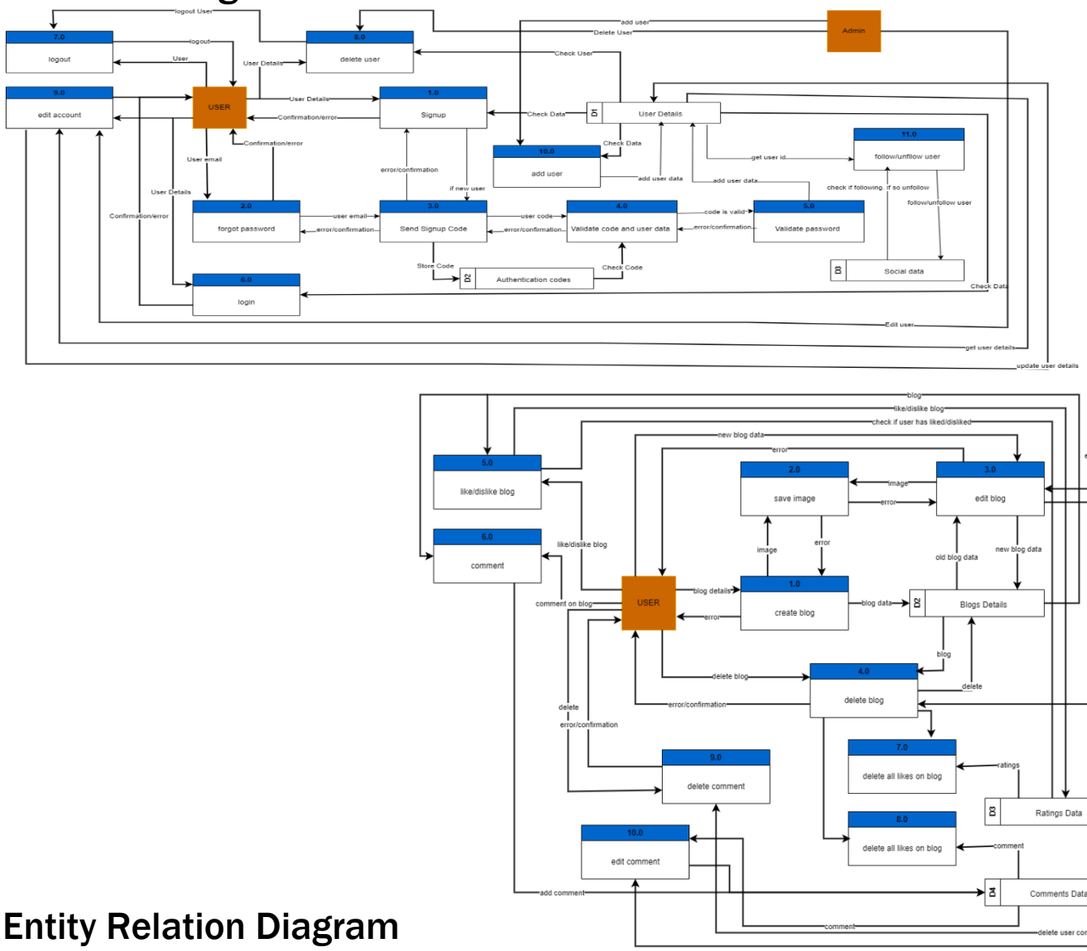
Passwords are hidden to improve the user's privacy and security.

Passwords will also be hashed upon account creation (SDCpc8)

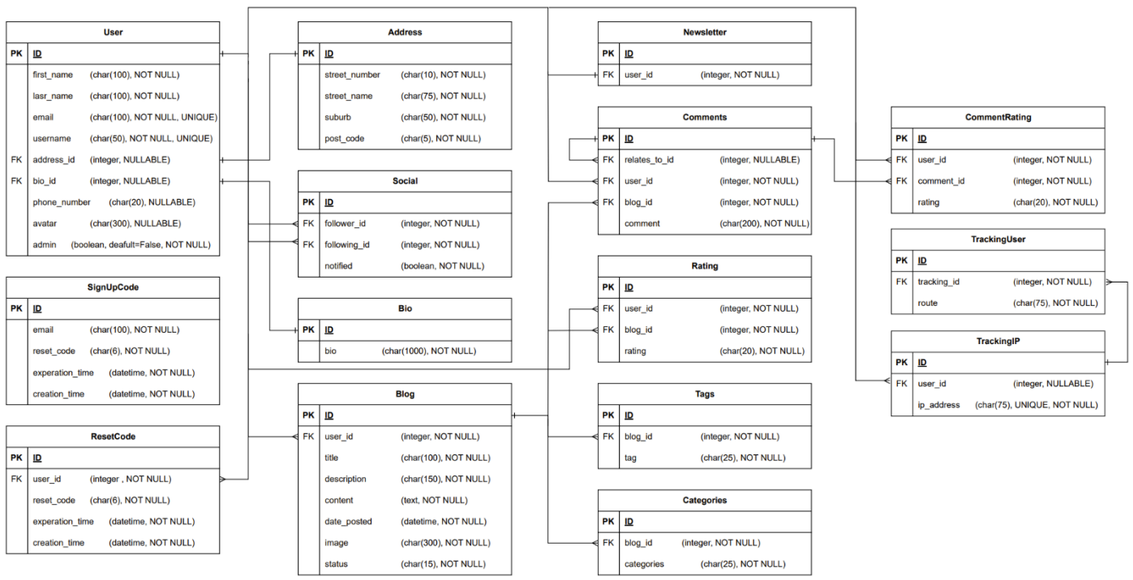
Data - Variable Table

Variable Name	Variable Description
blogs	Stores all blogs in a pagination list
db	The initialised database object for SQLAlchemy
app	The initialised variable for the flask app
Current_user	Holds the data for the current user
Related_comments	Stores nested comments for the blog that are related to other comments
User.admin	Is a Boolean for whether the user is an admin
form	The initialised object of the form
error	Stores the error for any forms
Main, signup, account, blog, ...	Stores the initialised object for the blueprint of the flask app
title	Stores the title for the webpage
matrix	Is a 2D list (matrix) of distances between two strings
suggestion	A suggested word based on the incorrect spelling of a word

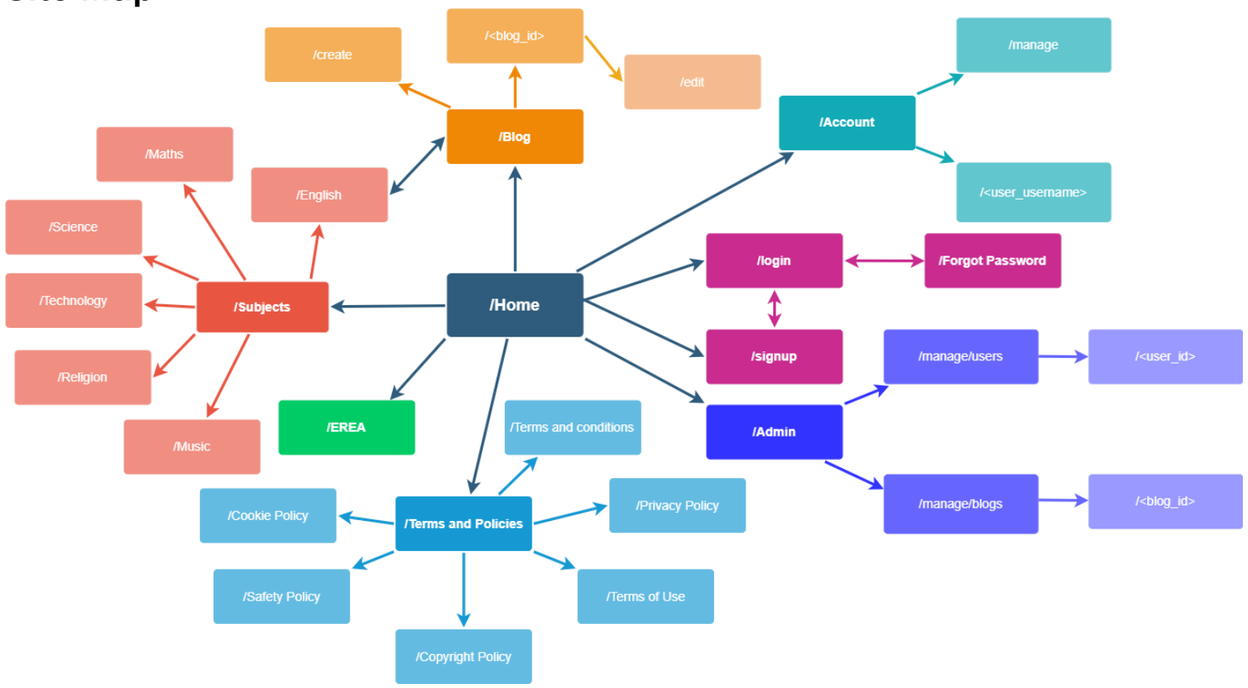
Dataflow Diagrams



Entity Relation Diagram



Site Map



Data Dictionaries

User Table

Field Name	Data Type	Data Format	Field Size	Description	Example
id	integer	Primary Key	32 digits	Is the primary key for the user table	3
first_name	string		100 chars	The user's first name	Nathan
last_name	string		100 chars	The user's last name	Perrier
email	string	Email address	100 chars	Is the primary identification of the user	056671@atc.qld.edu.au
username	string		50 chars	Is the users display name	Nathanperrier23
address_id	integer	Foreign Key	32 digits	The forging key for the address table	5
bio_id	integer	Foreign Key	32 digits	The foreign key for the bio table	2
phone_number	string		20 chars	The user's phone number	0412345678
avatar	string	directory	300 chars	The directory of the user's avatar	../d23g4f8asd3.png
admin	Boolean		1 digit	Stores where the user is an admin	1 (True)

Blog Table

Field Name	Data Type	Data Format	Field Size	Description	Example
id	integer	Primary Key	32 digits	Is the primary key for the user table	7
User_id	integer	Foreign Key	32 digits	The forging key for the user table	1
title	string		100 chars	The title of the blog	A blog about blogs
description	string		150 chars	The blog's description	Blogs are the panicle of...
content	datetime	DateTime	date	The date of when the blog was posted	12/12/23
image	string	directory	300 chars	The directory of the blog's image	../d23g4f8asd3.png
status	string		20 chars	The status of the blog	"active"

Pseudocode

Pseudocode

```
BEGIN
// Define the Levenshtein distance function
FUNCTION levenshtein_distance(self, s1, s2):
    // Create a matrix to store distances
    SET matrix = create a 2D list of zeros with dimensions (len(s1) + 1) x (len(s2) + 1)

    // Initialize the first row and column
    FOR i FROM 0 TO len(s1):
        SET matrix[i][0] = i

    FOR j FROM 0 TO len(s2):
        SET matrix[0][j] = j

    // Fill in the matrix using dynamic programming
    FOR i FROM 1 TO len(s1):
        FOR j FROM 1 TO len(s2):
            // Calculate the cost
            SET cost = 0 IF s1[i - 1] IS EQUAL TO s2[j - 1] ELSE 1

            // Update the current cell with the minimum of adjacent cells + cost
            SET matrix[i][j] = MIN(matrix[i - 1][j] + 1, matrix[i][j - 1] + 1, matrix[i - 1][j - 1] + cost)

    // Initialize pointers to the bottom-right of the matrix
    SET i, j = LENGTH(s1), LENGTH(s2)

    // Iterate over the matrix to find the edit operations
    WHILE i IS MORE THAN 0 OR j IS MORE THAN 0:
        IF i IS MORE THAN 0 AND j IS MORE THAN 0 AND s1[i - 1] IS EQUAL TO s2[j - 1]:
            // Characters are the same, move diagonally
            SET  i = i - 1
            SET  j = j - 1
        ELSE IF i IS MORE THAN 0 AND matrix[i][j] IS EQUAL TO matrix[i - 1][j] + 1:
            // Move up
            SET i = i - 1
        ELSE IF j IS MORE THAN 0 AND matrix[i][j] IS EQUAL TO matrix[i][j - 1] + 1:
            // Move left
            SET  j = j - 1
        ELSE:
            // Move diagonally
            SET i = I - 1
            SET j = j - 1

    // Return the shortest distance (bottom-right cell of the matrix)
    RETURN matrix[LENGTH(s1)][LENGTH(s2)]
END
```

```
BEGIN
// Define the class method to get featured blogs
@classmethod
FUNCTION get_featured_blogs(cls, db):
    // Get all ratings from the database
    SET ratings = call Rating().get_all(db)

    // Create dictionaries to store the total likes and dislikes for each blog
    SET blog_likes = {}
    SETblog_dislikes = {}

    // Calculate total likes and dislikes for each blog
    FOR EACH rating IN ratings:
        IF rating.blog_id NOT IN blog_likes:
            SET  blog_likes[rating.blog_id] = 0
        IF rating.blog_id NOT IN blog_dislikes:
            SET blog_dislikes[rating.blog_id] = 0

        IF rating.rating:
            SET blog_likes[rating.blog_id] += 1
        ELSE:
            SET blog_dislikes[rating.blog_id] += 1

    // Calculate the blog ratings based on likes and dislikes for each blog
    FOR EACH rating IN ratings:
        SET total_likes = blog_likes[rating.blog_id]
        SET total_dislikes = blog_dislikes[rating.blog_id]

        // Calculate the rating using the calculate_blog_rating method
        SET rating.v = call Rating().calculate_blog_rating(total_likes, total_dislikes)

    // Sort the ratings in descending order
    SET sorted_ratings = sort ratings by key=lambda r: r.v descending

    // Get the top 5 blog ids with the highest ratings
    SET top_blog_ids = [r.blog_id for r in sorted_ratings]
    SET top_blog_ids = top_blog_ids[:5]

    // Get the top 5 blogs with the highest ratings
    SET top_blogs = []
    FOR EACH blog_id IN top_blog_ids:
        top_blogs.append(call Blog().get_by_id(db, blog_id))

    RETURN top_blogs

// Define the method to calculate blog ratings
@classmethod
FUNCTION calculate_blog_rating(self, likes, dislikes):
    // Calculate the ratio of likes to total votes (likes + dislikes)
    SET  total_votes = likes + dislikes

    // Calculate the activity factor
    SET  activity_factor = 1 + min(1, total_votes / 50)

    // Calculate the ratio and map it to the range [0, 2π]
    SET  ratio = (0 IF total_votes == 0 OR likes == 0 ELSE likes / total_votes)
    SET  x = 2 * math.pi * ratio

    // Calculate the rating using the given function
    SET rating = ((2.5 * math.cos(x) + 2.5) IF x > 0 ELSE 0) * activity_factor

    // Round the rating to a maximum of 5 with a minimum of 0
    RETURN round(min(5, max(0, rating)), 2)
END
```

```
BEGIN
// Define a class method to create a new blog
@classmethod
FUNCTION create_blog(cls, db, user_id, title, description, content, image=None):
    TRY:
        // Check if an image is provided; if not, generate one using AI
        IF image IS None:
            SET image = ImageGeneration().get_ai_image(description)
        ELSE:
            SET image = cls.save_image(image, description)

        // Create a new blog object with the provided data
        SET blog = create a new instance of cls with user_id, title, description, content, and image

        // Add the blog to the database session and commit changes
        db.session.add(blog)
        db.session.commit()

        // Return success status, an empty error message, and the ID of the created blog
        RETURN True, '', blog.id

    EXCEPT Exception AS e:
        // Handle exceptions and return failure status along with the error message and an empty ID
        RETURN False, str(e), ''

// Define a class method to edit an existing blog
@classmethod
FUNCTION edit_blog(cls, db, id, title, description, content, image):
    TRY:
        // Retrieve the blog by its ID
        SET blog = cls.get_by_id(db, id)

        // Update the blog's attributes with the provided data
        SET blog.title = title
        SET blog.description = description
        SET blog.content = content
        SET blog.image = image

        // Commit the changes to the database
        db.session.commit()

        // Return success status and an empty error message
        RETURN True, ''

    EXCEPT Exception AS e:
        // Handle exceptions and return failure status along with the error message
        RETURN False, str(e)
```

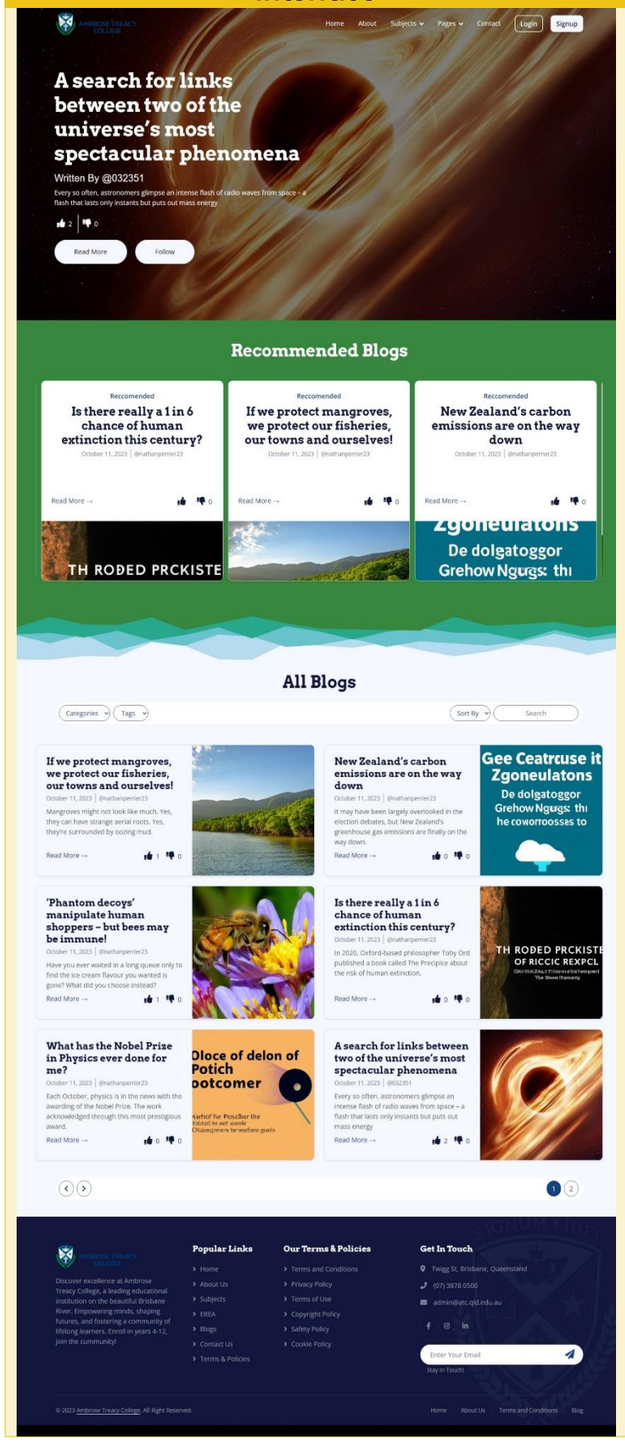
```
// Define a class method to retrieve a blog by its ID
@classmethod
FUNCTION get_by_id(cls, db, id):
    // Query the database to find the blog with the specified ID
    RETURN db.session.query(cls).filter_by(id=id).first()
END

BEGIN
// Import necessary libraries and modules
FROM flask IMPORT Flask, render template, request, jsonify
FROM flask_login IMPORT login_required, current_user
FROM your_app.models IMPORT Blog

// Define a Flask Blueprint for the 'blog' module
SET blog = create a Flask Blueprint for 'blog' with __name__ as 'blog'

// Define the route to create a blog page
@blog.route('/blog/create', methods=['GET', 'POST'])
@login_required
FUNCTION create_blog_page():
    // Render the 'create.html' template with title and user information
    RETURN render_template('blogs/create.html', title=TITLE, user=current_user)

// Define the route to post a new blog
@blog.route('/blog/create/post', methods=['POST'])
@login_required
FUNCTION post_blog():
    TRY:
        // Try to get an image from the request
        SET image = get the 'image' file from request
    EXCEPT:
        // If no image is provided, set it to None
        SET image = None

    // Create a new blog using the Blog model
    SET valid, error, blog_id = Blog().create_blog(db, current_user.id, request.form.get('title'), request.form.get('description'), request.form.get('content'), image)

    // Return a JSON response with success status, error message, and blog ID
    RETURN jsonify(success=valid, error=error, id=blog_id)

// Define the route to edit a blog page
@blog.route('/blog/<int:blog_id>/edit', methods=['GET', 'POST'])
@login_required
FUNCTION edit_blog_page(blog_id):
    // Get the blog by its ID using the Blog model
    SET blog = Blog.get_by_id(db, blog_id)

    // Render the 'edit.html' template with title, user, and blog information
    RETURN render_template('blogs/edit.html', title=TITLE, user=current_user, blog=blog)

// Define the route to post edited blog content
@blog.route('/blog/<int:blog_id>/edit/post', methods=['POST'])
@login_required
FUNCTION post_blog_edit(blog_id):
    TRY:
        // Try to get an image from the request
        SET image = Blog().save_image(request.files['image'], request.form.get('description')) IF request.form.get('image') IS None ELSE request.form.get('image')
    EXCEPT:
        SET image = None

    // Edit the existing blog using the Blog model
    SET valid, error = Blog().edit_blog(db, blog_id, request.form.get('title'), request.form.get('description'), request.form.get('content'), image)

    // Return a JSON response with success status and error message
    RETURN jsonify(success=valid, error=error)

// Define the route to delete a blog
@blog.route('/blog/<int:blog_id>/delete', methods=['POST'])
@login_required
FUNCTION delete_blog(blog_id):
    // Delete the blog using the Blog model
    SET success, error = Blog().delete_blog(db, blog_id)

    // Return success status and error message
    RETURN success, error
```

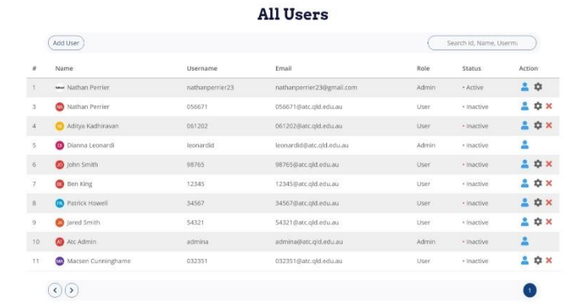
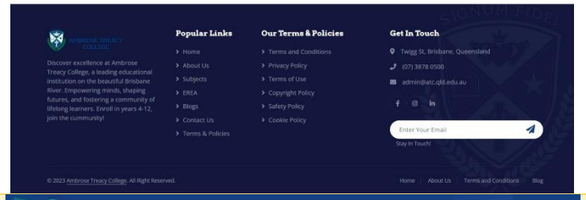
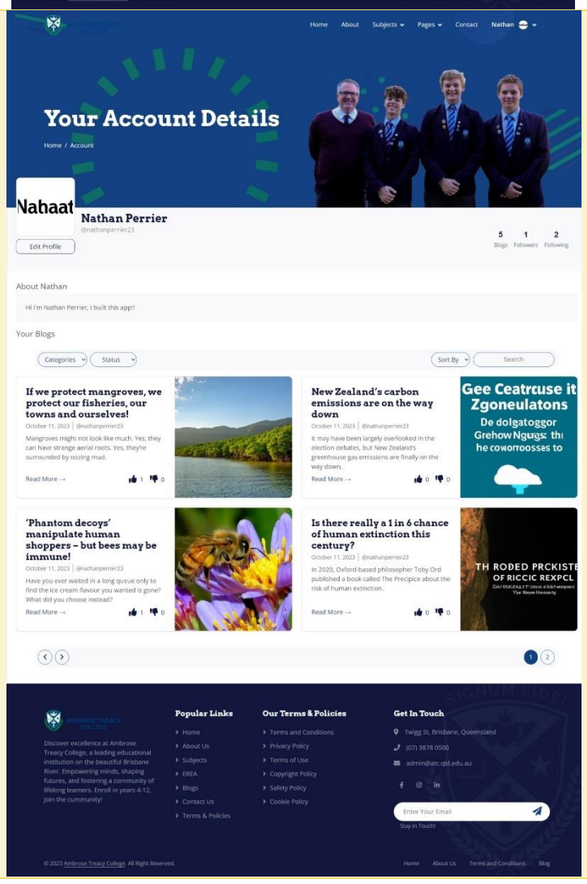
```
@classmethod
FUNCTION check_email(cls, email):
    """Check if the email is in the database."""
    result = db.session.query(User).filter_by(email=email).first()
    RETURN result
```

```
@classmethod
FUNCTION is_name_valid(cls, name):
    """Check if the name is valid"""
    valid_chars = "ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz-"
    FOR char IN name:
        IF char NOT IN valid_chars:
            RETURN False
    RETURN True

@staticmethod
FUNCTION generate_code(length=6):
    """Generate a random numerical reset code of the given length."""
    code = ''
    FOR _ IN RANGE(length):
        random_char = random.choice(string.ascii_letters + string.digits)
        code += random_char
    RETURN code

@staticmethod
FUNCTION send_reset_code(first_name, email, code):
    """Send the reset code to the user's email."""
    TRY:
        send_email('contact.webgenieai@gmail.com', email, create_email(first_name, email, 'Verification Code', code))
    EXCEPT Exception AS e:
        PRINT(e)

@classmethod
FUNCTION store_reset_code(cls, email, code, expiration_duration=1800): # 30 mins
    """Store the reset code in the database with an expiration time."""
    expiration_time = datetime.utcnow() + timedelta(seconds=expiration_duration)
    reset_entry = SignupCode(email=email, reset_code=generate_password_hash(code), expiration_time=expiration_time, creation_time=datetime.utcnow())
    db.session.add(reset_entry)
    db.session.commit()

@classmethod
FUNCTION create_and_send_reset_code(cls, first_name, last_name, email):
    """Generate, send, and store the reset code."""
    email_check = cls.check_email(email)
    IF email_check IS None:
        IF cls.is_name_valid(first_name) AND cls.is_name_valid(last_name):  // check if the user's name valid
            IF cls.check_email_format(email):  //check if email is correct format
                IF cls.check_active_code(email):  // check if code is active
                    // if there is no active code
                    code = cls.generate_code()
                    cls.send_reset_code(first_name, email, code)
                    cls.store_reset_code(email, code)
                    RETURN True, None
                RETURN True, None    // has active code (continue)
            RETURN False, 'Invalid Email Format'
        RETURN False, 'Invalid Name(s)'
    RETURN False, "Email Already Registered"

@classmethod
FUNCTION check_email_format(cls, email):
    // checks if email has an @ and a .
    TRY:
        username, domain = email.split('@')
        domain, tld = domain.rsplit('.', 1)
        RETURN True
    EXCEPT ValueError:
        RETURN False

@classmethod
FUNCTION check_active_code(cls, email):
    // checks if there is a code that's experation time has not been reached
    active_code_check = db.session.query(cls).filter_by(email=email).filter(SignupCode.expiration_time > datetime.utcnow()).first()
    IF active_code_check:
        RETURN False
    RETURN True

@classmethod
FUNCTION check_code(cls, user_code, code):
    // unhashed code and checks if the inputted code is the same
    RETURN check_password_hash(user_code, code)

@classmethod
FUNCTION check_reset_code(cls, email, input_code):
    // checks if code is valid
    reset_entry = db.session.query(cls).filter_by(email=email).order_by(SignupCode.creation_time.desc()).first()

    IF NOT reset_entry:
        RETURN False, "No reset code found for the user"

    IF cls.check_code(reset_entry.reset_code, input_code) IS False:
        RETURN False, "Invalid reset code"

    current_time = datetime.utcnow()
    IF current_time > reset_entry.expiration_time:
        RETURN False, "Reset code has expired"

    RETURN True, None
```

Generate

User Interfaces

Interface	Annotations
	Good use of white-space to ensure the webpage is not “crowded” with information or data Has a slideshow to maximise the number of featured blogs that can be displayed Blog images act as a background image for the slideshow Readers can press on “read more” to be redirtected to blog page. This is to increase the utility The featured blogs are gathered from the number of likes a blog has The recommended blogs are also displayed on a slideshow. Users can click on them to be read more An animation occurs when hovered on the recommended section There is a filter section where users could filter or search blogs, however this feature was not added. The layout and styling is neat, modern and not overwhelming. Followiungs the clients scheme A max of 6 blogs is displayed at a time. Users can use the navbar at the bottom of the section to change the page The responsiveness of this page will be adjusted to all screen sizes to improve accessibility (SDC14) All buttons and navbars are located in logical locations to improve the learnability of the site The contrst of text and colours is reasonable to improve accessability Users can follow/unfollow user’s whose blogs are displayed on the featured section. If the current user is an admin then it will be a edit button.

  	User’s passwords are not displayed to increase the sites safety All important information is displayed for each user. The page can only be accessed by admins. If theuser is not an admin the will be redirected to a 401 (unauthorised) error page. Admins can add a new user without needed to receive a authentication code Admins can search users by id, username, email or name. based on the search users willl be sorted in terms of similarity. Admins can veiw, edit or delete users. If the user is an admin then they can only veiw the account. Also admins can only veiw or edit their own account, they cannot delete their own account. The table uses pagination to minise the amount of verticle space needed. Only 10 users are displayed each page. The table fields are evenly space. Furthermore the table is responsive to improve accessbility. The layout of the table is simple to improve leanrability. Furthermore, the symbols used for the buttons are intuative The profile picture of each user is displayed.
	This design was heavily inspired by the existing solution analysed. As can be see the interface takes multiple elements from the analysed design. This was done to improve learnability as the design and layout is popular. Both the your account page and user account page use a similar layout The user’s profile picture is displayed, with a edit account button bellow. The user’s name and username are displayed at the top of the page to make it easy to see. the number of blogs, follower and following is displayed to inform the user the user’s bio is displayed on this page. The bio can be updated in the manage account page the user’s blogs are also displayed here, that visitors to their page will also see. The blogs have filter ovtions in the top navbar, however this was not fully implimented. While only 4 blogs per page are displayed, users can cycle through their blogs usning the bottom page navbar. This page is fully respoonsive to improve accessiblity. Furhtermore, users casn use the footer for additional navigation.

Code

Code

```
def levenshtein_distance(self, s1, s2):
        # Creates a matrix to store distances
        matrix = [[0 for _ in range(len(s2) + 1)] for _ in range(len(s1) + 1)]  # creates
a 2D list (matrix) of zeros with dimensions (len(s1) + 1) x (len(s2) + 1)

        # Initialize the first row and column
        for i in range(len(s1) + 1):
            matrix[i][0] = i # fill in the first column with range(len(s1) + 1)
        for j in range(len(s2) + 1):
            matrix[0][j] = j # fill in the first row with range(len(s2) + 1)

        # Fill in the matrix using dynamic programming
        for i in range(1, len(s1) + 1):
            for j in range(1, len(s2) + 1):
                cost = 0 if s1[i - 1] == s2[j - 1] else 1 # cost is 0 if the characters
are the same, 1 otherwise
                matrix[i][j] = min(matrix[i - 1][j] + 1, matrix[i][j - 1] + 1, matrix[i -
1][j - 1] + cost) # fill in the current cell with the minimum of the three adjacent cells
+ cost

        i, j = len(s1), len(s2)
        while i > 0 or j > 0: #itierates over the matrix until it reaches the top left
corner
            if i > 0 and j > 0 and s1[i - 1] == s2[j - 1]: #if the characters are the
same, move diagonally
                i -= 1  #move diagonally
                j -= 1
            elif i > 0 and matrix[i][j] == matrix[i - 1][j] + 1: #if the cost of the
current cell is equal to the cost of the cell above + 1, move up
                i -= 1 #move up
            elif j > 0 and matrix[i][j] == matrix[i][j - 1] + 1: #if the cost of the
current cell is equal to the cost of the cell to the left + 1, move left
                j -= 1 #move left
            else:
                i -= 1 #move diagonally
                j -= 1

        return matrix[len(s1)][len(s2)]
```

```
# From: backend/hosting/__init__.py
from backend.config import *
from backend.__init__ import *

def create_app():
    # Initialize app
    app = Flask(__name__, template_folder='../../frontend/templates/',
static_folder='../../frontend/static/')
    # Configure app
    app.config['SQLALCHEMY_DATABASE_URI'] = 'sqlite:///site.db'
    app.config['SECRET_KEY'] = random.randbytes(32)
    app.config['BLOG_UPLOAD_FOLDER'] = 'frontend/static/images/user-images/blogs/'
    app.config['AVATAR_UPLOAD_FOLDER'] = 'frontend/static/images/user-images/avatars/'
    app.config['AI_UPLOAD_FOLDER'] = 'frontend/static/images/ai-images/blogs/'

    # Initialize extensions
    login_manager.init_app(app)

    # Register blueprints
    app.register_blueprint(main)
    app.register_blueprint(sign_up)
    app.register_blueprint(terms)
    app.register_blueprint(blog)
    app.register_blueprint(subject)
    app.register_blueprint(forgot)
    app.register_blueprint(account)
    app.register_blueprint(admin)
    app.register_blueprint(tracking)

    # Register error handlers
    register_error_handlers(app)

    # Initialize database
    db.init_app(app)

    # Create database tables
    with app.app_context():
        db.create_all()

    return app
```

```
def get_ai_image(self, description):
        # get the response from the OpenAI API
        response = openai.Image.create(
            prompt="Generate a digital image for a blog post with the following
description: " + description,
            n=1,
            size="1024x1024",
        )
        # save the image to the server
        name = self.save_image(response['data'][0]['url'])
        return FILE_PATH + name

def save_image(self, url, file_path=None):
        # get image unique directory
        directory = self.check_dir((FILE_PATH_AVATAR + self.generate_name() + '.png'),
file_path=True) if file_path else self.check_dir((FILE_PATH + self.generate_name() +
'.png'))
```

```
        # get url data
        img_data = requests.get(url).content
        # save image to directory
        with open(directory, 'wb') as handler:
            handler.write(img_data)
        if file_path:
            return directory.split(FILE_PATH_AVATAR)[1]
        return directory.split(FILE_PATH)[1]

    def generate_name(self, length=50):
      # generate a random name for the image
        return ''.join(random.choice(string.ascii_letters + string.digits) for _ in
range(length))

    def check_dir(self, directory, file_path=None):
        # check if the directory exists
        if os.path.exists(directory):
            return self.check_dir((FILE_PATH + self.generate_name() + '.png')) if
file_path == None else self.check_dir((FILE_PATH_AVATAR + self.generate_name() + '.png'),
file_path=True)
        return directory
```

```
@sign_up.route('/signup', methods=['GET', 'POST'])
def signup_page(error=''):
    return render_template('signup.html', title=TITLE, error=error)

@sign_up.route('/signup/validate_email', methods=['POST'])   #! issue
def signup_validate_email():
    valid, error = SignupCode.create_and_send_reset_code(request.form.get('first_name'),
request.form.get('last_name'), request.form.get('email'))
    return jsonify(success=valid, error=error)

@sign_up.route('/signup/validate_code', methods=['POST'])
def signup_validate_code():
    valid, error = SignupCode.check_reset_code(request.form.get('email'),
request.form.get('reset_code'))
    return jsonify(success=valid, error=error)

@sign_up.route('/signup/validate_password', methods=['POST'])
def signup_validate_password():
    valid, error =
User.signup(request.form.get('first_name'),request.form.get('last_name'),
request.form.get('email'), request.form.get('new_password'))
    return jsonify(success=valid, error=error)
```

```
from backend.config import *
from backend.tests.db import db

class TestFlaskApp(unittest.TestCase):
```

```
    def setUp(self):
        # initialize the flask app
        self.app = Flask(__name__)
        self.app.config['TESTING'] = True
        self.client = self.app.test_client()

    def test_index(self):
        # test the index page
        response = self.client.get("/")
        self.assertEqual(response.status_code, 200)
        self.assertIn(b"Welcome to the Tracking App", response.data)

    def test_404(self):
        # test the 404 page
        response = self.client.get('/404')
        self.assertEqual(response.status_code, 404)
        self.assertIn(b"Page not found", response.data)

    def test_500(self):
        # test the 500 page
        response = self.client.get('/500')
        self.assertEqual(response.status_code, 500)
        self.assertIn(b"Internal server error", response.data)

    def test_login(self):
        # test the login page
        response = self.client.get('/login')
        self.assertEqual(response.status_code, 200)
        self.assertIn(b"Login", response.data)

    def test_signup(self):
        # test the signup page
        response = self.client.get('/signup')
        self.assertEqual(response.status_code, 200)
        self.assertIn(b"Register", response.data)
```

Evaluation

Testing

TEST	EXPECTED OUTCOME	ACTUAL OUTCOME	RECOMMENDATIONS	ACHIEVED
A Test to use dynamic ids for comment inputs	JavaScript would retrieve the input data using the dynamic ids	The data would not be retrieved	Check if there are ids with the same name	Yes, had double ups of id names so was not finding correct id
A test to use JavaScript to post user data	JavaScript would post data to python route	Would raise an error about blog_id being missing	Pass through blog_id in route	Yes, included blog_id in route using jinja
A test to use pagination	Would allow for all users to be displayed over many pages	Arrows would not redirect to correct page	Make sure the correct page number is being passed through	Yes, was not passing through correct page number
A test to store user data	Would store the new user	User would be deleted on restart	Make sure data is being committed	Yes, added db.session.commit()

User Testing

Type of Test	Completion Status
Can the user open the website?	[✓]
Can the user close the website?	[✓]
Can the user navigate the website?	[✓]
Can the user login?	[✓]
Can the user logout?	[✓]
Can a user get admin permissions (if applicable)?	[✓]
Can the user edit/delete their data?	[✓]
Can the admin edit/delete user and blog data?	[✓]
Can the user create a new account?	[✓]
Can the user create/edit/delete a blog?	[✓]
Can the user comment on blogs?	[✓]
Can the user like/dislike blogs?	[✓]
Can the user follow/unfollow other users?	[✓]

User Feedback

Name	Feedback
Jasmine	"Home screen is welcoming. Great inclusion of footage of the school and text/font is fitting the professional aesthetic of the website. Contact us section is fully functional and easy to use. Some buttons on homepage don't do anything. Well-presented pages that have a lot of variety. Good design in the layout of the blog posts. Search bar within the blog page is not yet functional. Creating a blog was an easy process, steps were clearly illustrated, and all aspects were functional. I like the admin page and the fact that I can search users"
Aditya	"When testing the website, I was pleasantly surprised of how well the html and CSS was designed. The login system was top notch and could be compared with other competitors. The only issues I faced with the website were the slow animations. Alse Emails do not send when on school Wi-Fi"

Criteria

PRESCRIBED CRITERIA			SELF-DETERMINED CRITERIA		
PC1	Develop a website with many pages and features	[✓]	SDC1	Using objects to code in a modular way	[✓]
PC2	Allow multiple users to use the website and have the ability for new users to be added	[✓]	SDC2	Using lists, arrays, and dictionaries to store data	[✓]
PC3	Use databases for data storage	[✓]	SDC3	Allow users to upload and customise their profile picture	[✓]
PC4	Create a folio that outlines the user interface and coded components of the solution	[✓]	SDC4	Use authentication codes to confirm user authenticity	[✓]
PC5	Have blog functionality that all users, can use, view and access	[✓]	SDC5	Send emails to users to send necessary information	[✓]
PC6	Generate a user/visual interface	[✓]	SDC6	Use JS for data handling and design elements	[✓]
PC7	Incorporate the use of the client's signature colours or logo	[✓]	SDC7	Allow the users to reset their passwords if forgotten	[✓]
PC8	Integrate security features such as hashing and encryption	[✓]	SDC8	Develop and use the Lowenstein distance algorithm	[✓]
PC9	Users must be able to edit or delete their information	[✓]	SDC9	Integrate a spell check in the blog functionality	[X]
PC10	Have admin functionality that is given to the client's staff. They should be able to edit, delete and add user and blog data.	[✓]	SDC10	Have nested comments for blog that users can edit and delete	[✓]
PC11	Allow for users to login, signup and logout	[✓]	SDC11	Use SQLAlchemy and a multi-file architecture for the code design	[✓]

CRITERIA	REASON
SDC9	This feature was not added due to a lack of time and insufficient training data
SDC18	This feature was not added as there was not enough time
SDC19	While the code was developed, the training data was not sufficient
SDC20	There was not enough time to implement this as the frontend part of it required a bit of time
SDC21	This feature was not added as it was mainly needed for the AI features that were not added
SDC22	This was not added as the algorithm to do this is quite complex
SDC24	As this featured required for the database and its process to be reconfigured, it became too time consuming
SDC31	This feature required tracking to be added first, which is why it was not implemented
SDC32	This feature required to much time to add

SDC12	Use CSS to style the website to adhere closely with the client's style	[✓]
SDC13	Use animations on the website to engage the user	[✓]
SDC14	Prioritise website responsiveness to improve accessibility	[✓]
SDC15	Have a like-dislike system for blogs	[✓]
SDC16	Use AI to generate blog images if the user has no image to upload	[✓]
SDC17	Have a social system where users can follow/unfollow other users	[✓]
SDC18	Develop a notification system where user's will be notified when they are followed or a user, they are following uploads a blog	[X]
SDC19	Have an AI to suggest words or sentences as the user is typing	[/]
SDC20	Have an AI to generate blog titles or descriptions	[X]
SDC21	Allow for users to add tags or categories to their blogs to allow to inform readers and add context. This will also be used for the search feature	[X]
SDC22	Allow users to search for blogs or users	[X]
SDC23	Have a featured blog section were the blogs with the most likes or engagement (number of comments) will appear	[✓]
SDC24	Allow users to achieve their blog posts or save as a draft when creating one	[X]
SDC25	Use drone footage of the Client's campus to engage users when they click on the website	[✓]
SDC26	Have terms and policies regarding the use of the website. Make users agree to these terms before signing up	[✓]
SDC27	Have multiple pages regarding information about the client such as subjects or their relation to EREA	[✓]
SDC28	Have a contact form where users can ask questions	[✓]
SDC29	Have a newsletter where users can sign up and receive newsletters	[✓]
SDC30	Have unit tests for various programming elements	[✓]
SDC31	Have activity logs that the user and admin can view	[X]
SDC32	Have tracking functionality that identifies the user's or a device's likes in terms of the blogs they view. Use this data to recommend blogs to the user.	[X]
SDC33	Have AI generated and default profile pictures gathered from API's.	[✓]
SDC34	Allow admins to search for users based on their id, username, name or email	[✓]

Impacts

Social

As this project is a social app, it introduces many social impacts that need to be considered. As the success criteria outlines many features that bring users together, such as following/followers (SDC17), likes/dislikes (SDC15), comments (SDC10), notifications (SDC18), and blogging (PC5). These features make the app allow communities to connect with one another. Furthermore, due to the inclusion of user's being able to communicate with others through the comments and blogging system, users can plan or advertise social events. Finally, while users can voice their opinions, which may be harmful to other users. while this is addressed in the app's terms and conditions (SDC28), the impact is still present.

Personal

A personal impact due to the app's development is the fact that users could get addicted to the app and its features. This means that users could prioritise the use of the app over other aspects of their life, such as going outside. Finally, as partially mentioned previously, users on the app could encounter harassment or bullying through the app's communication system. This may lead to potential mental health issue that pose as a vast personal impact. Finally, through the use of AI it could pose as a personal threat of data security.

Economical

Through the use of the app's communication and blogging system, businesses or products could be advertised that would benefit the economy. Furthermore, with the possible expansion of this app, employees could be hired to aid with the development. This would introduce many new jobs in the local area. Furthermore, to compensate for the cost of employees and future politization, monetisation could be used to generate a sufficient income. The app could be monetised through, adds, subscriptions or an on-time-fee.

Legal

As the app utilises AI in the aid of content creation, there is various legal impacts in its use. As AI could leak, or misuse user data its user has to be monitored. Furthermore, to cover the apps and client's liability in the use of AI, the user upon signup must agree to the site's terms and conditions (SDC28). Furthermore, the AI and the app's users could spread misinformation, where legal action may be necessary. A major legal liability is the applications use in the conduction of illegal activities. Finally, legal action may be needed if a person or group is harmful to the app or its users.

Usability recommendation

As usability was a key part of the design of the website there are little improvements to be made. Although after extensive user feedback it was determined that there are some features and buttons that are not function. As time was a major restriction in the app's development, some pages and buttons found on the homepage and other various pages were not function and are purely decorative. Furthermore, as outlined in the success criteria there are multiple features that could not be completed. This resulted in many buttons, inputs and sections being incomplete in the blogging, user and account sections. Furthermore, it was identified that the emailing library used does not work when on a organisations Wi-Fi network. While the library could be swapped, a private network can be used to access the feature. Furthermore, it was identified that the clear but throughout inclusion of the clients' colours, pictures and logos was commended. Additionally, users found all forms and navigation easy to use and follow. However, it was identified that the animations and main video were slow, this is mainly due to hardware limitations, but adjustments could be made. Overall, the usability feedback on the website was minimal with only the missing features being the main issue.

Accessibility Checklist

Accessibility	Achieved
Text is large enough to easily read	[✓]
User can exit or logout at anytime	[✓]
Do all buttons text boxes work	[✓]
Information is provided if the user would ever be unsure of what to do	[✓]
Information is provided is an error occurs	[✓]
Text colour contrast is reasonable	[✓]
Can a screen reader describe all images used	[✓]
Is audio available for users with hearing-impairment	[✓]

Additional Evaluation

Risks Taken

During planning and development of this project it was identified that the scale was a major risk due to the limitation of time. As seen in the success criteria there are 34 self-determined criteria's, this required for vast amounts of time and skill to be required. While not all of these self-determined criteria were successfully developed, over 27 were included. Furthermore, on top of this, 11 prescribed criteria were necessary, resulting in a total of 38 implemented features. Therefore, the scale of project caused around 20,000 lines of code, over 100,000 files and 7GB of storage to produce the final product. Thus, displaying the significant risk taken in implementing the success criteria.

Furthermore, there are certain features presented in the success criteria that introduced risk in the completion of the task. To add store the data for all the app processes, SQLAlchemy (SDC11) was used as it allows for query-less altering. Thus, to implement the database the code had to be structured to what was required. This meant that, as I was using Flask, Flask blueprints were needed. Flask blueprints allow for multiple hosting files to be used, this made it possible to use SQLAlchemy without any circular import errors. However, the implementation of these two features took a vast amount of time. Additionally, as the design of the website was an integral part of the project, a lot of time was dedicated to the creating, refining and implementing many of the webpages (SDC12). Furthermore, to excel the web design of the project, various animations we used to make the website more entertaining (SDC13). However, to include these animations, knowledge in the use of API's, Plugin's and JavaScript was needed. Further knowledge in JavaScript was needed in order to manage the form processes, this is because it was used primarily to retrieve data, process data and post data. Therefore, time was necessary in gaining the required skill to complete this.

As some features within the success criteria required extensive skill and complexity, it made implementing these features a vast risk due to the time required. To increase the safety and security of the app, code authentication was added to the signup and forgot password forms (SDC4). This was done to identify if the email entered is in fact a valid email. To add this feature, two addition database tables and multiple functions had to be added. Moreover, to increase the social aspects of the website, comments (SDC10), likes/dislikes (SDC15), follow/unfollow (SDC17) systems were added. These features, however, took a lot of time to add, making them a risk in the project's development. In addition, various AI models were added to ease aspects in data create for the users. To implement these models, a lot of time had to be dedicated into research and development to ensure the best possible result. Furthermore, an increase in skill and knowledge in this field had to be gained. While not all models were successfully implemented, these features required immense amounts of time. Finally, while this was not stated in the success criteria due to length constraints, the use of pagination on the user, admin and blog pages were a risk due to the time take to learn how to incorporate it. However, while it did take time to

add, it successfully displays users and blogs over multiple pages, to increase the overall design of the website.

While these are only some of the risks taken, it illustrates how due to the scale and complexity of this project, time became a major risk.

What Was Learnt

When developing the project there were many things that had to be learnt. The most challenging and time-consuming concepts were related to the AI algorithms. When attempting to develop and implement the spell checker and autocomplete features, the Levenstein algorithm (SDC8) had to be used. The Levenstein algorithm is used to calculate the similarity or distance between two strings (GeekForGeeks (b), 2023). As this algorithm requires extensive knowledge in matrixes and dynamic programming. Dynamic programming is mainly an optimization over plain recursion and is known to be a complex concept in programming (GeekForGeeks (a), 2023). With the added default of including matrixes, it made this algorithm very difficult to implement, however, it was successfully added.

Another feature that had a steep learning curve was the inclusion of text-completion algorithms (SDC19). While a working prototype of this feature was created, it was deemed insufficient for what was required. Even though this feature was not added, to get a working prototype, many stages of research and development was necessary. From the user of libraries to the development of original AI large language models, it required a vast number of concepts to be learnt.

Finally, the last feature that required a lot of knowledge was the implementation of recommended and featured blogs (SDC32) (SDC23). For the recommended blogs feature, it was decided that user tracking was to be used to understand what blogs the user likes. Using this data blogs would be recommended using AI models. While a prototype of the AI model was developed, it was not satisfactory for what was needed. Furthermore, due to time constraints the tracking feature was not implemented into the final product. While this feature was not complete it did require a lot of knowledge about user tracking and AI algorithms. To gain this knowledge, research into infamous recommendations algorithms was done. Such algorithms were created by companies such as twitter or LinkedIn. Furthermore, for the featured blogs, research was conducted to find the best possible solution. Eventually, it was concluded that a trigonometric function was ideal. This is due to the fact that it can provide a rating between 0 and 5 based on the derivative $\frac{dy}{dx}$ of the linear function (y=mx+c). Therefore, the following function was developed:

$$rating.q = f(x) = \begin{cases} 0, & x = 0 \\ \left(\frac{5}{2}\cos(2\pi(x) + \frac{5}{2}\right) \times (1 + \min(1, (total/50)), & x > 0 \end{cases}$$

$$\text{Where } x = \frac{dy}{dx}mx + c = \frac{likes}{total} \text{ and } total = likes + dislikes$$

Therefore, it can be shown that to develop these features, immense amounts of knowledge and research was necessary.

Improvements and Limitations

As time was a major limiting factor in this project, it meant that several successes criteria's were not implemented in the final product. Furthermore, additional implementations such as another admin page for user and blog tracking and various blogging features could have been added with additional time. These blogging features could have included the ability to have paragraph breaks, images in the content, intext-refencing and sub-headers. Another limitation was with the hardware. This is because, as the app used various animations and AI models it required a lot of computing power. This meant that the website may freeze due to the animations. Furthermore, it limited the use of AI as the use of original models were to much, thus, APIs had to be used instead.

Real-world Applications

The real-world applications for this project are vast, this is because many modern websites utilise many of the features found in the final product. With the addition of authentication codes to the signup and forgot password systems, it mimics the industry standard for secure authorisation (Wallace, 2021). Furthermore, the inclusion of the blogging system provides insight into many systems and features used on most of the social media platforms. Finally, the use of AI is a popular addition of most modern websites, demonstrating its significant real-world application in future development.

References

Aastha. (2021, July 20). *Object-oriented programming concepts you must know to be a software developer*. Medium. https://medium.datadriveninvestor.com/object-oriented-programming-concepts-you-must-know-to-be-a-software-developer-5bf16956f8cc

Applications of the 20 Most Popular Graph Algorithms. (2022, March 11). Applications of the 20 Most Popular Graph Algorithms. https://memgraph.com/blog/graph-algorithms-applications#:~:text=A*%20is%20used%20in%20many,best%20route%20between%20two%20nodes

bwasti. (n.d.). jott - py3.11_vs_3.8. Jott - py3.11_vs_3.8. Retrieved February 20, 2023, from https://jott.live/markdown/py3.11_vs_3.8#:~:text=It's%20been%20making%20strides%20in,noticable%20speedup%20on%20my%20system.&text=Python%203.8%20took%2096.79%20seconds.&text=Python%203.11%20took%20only%2031.98,That's%203x%20faster!

GeekForGeeks (a). (2023, September 26). Dynamic Programming. GeeksforGeeks. https://www.geeksforgeeks.org/dynamic-programming/

GeeksForGeeks (b). (2023, October 1). Introduction to levenshtein distance. GeeksforGeeks. https://www.geeksforgeeks.org/introduction-to-levenshtein-distance/

Olawanle. (2022, October 5). Big O Cheat Sheet – Time Complexity Chart. freeCodeCamp.org. Retrieved February 27, 2023, from https://www.freecodecamp.org/news/big-o-cheat-sheet-time-complexity-chart/

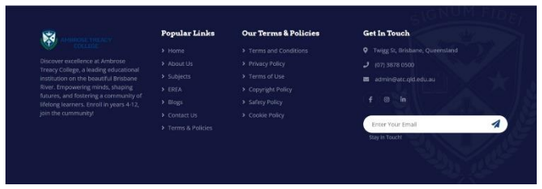
Shaw. (2018, July 16). Medium. Medium. Retrieved February 17, 2023, from https://medium.com/p/e5074b6fe55bone%20of%20the,interpreted%20languages%20such%20as%20JavaScript.

Wallace, B. (2021, August 5). What is Authentication & Why is it important?. Dumb Little Man. https://www.dumblittleman.com/why-is-authentication-important/

Appendices

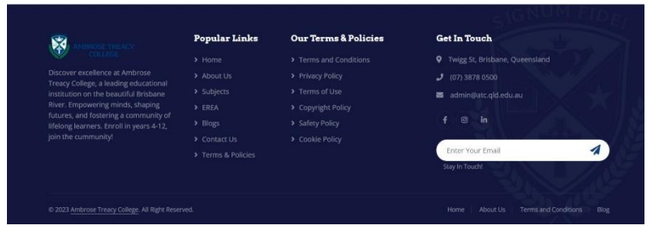

Home About Subjects Pages Contact John
401 Error
Home / 401 Error

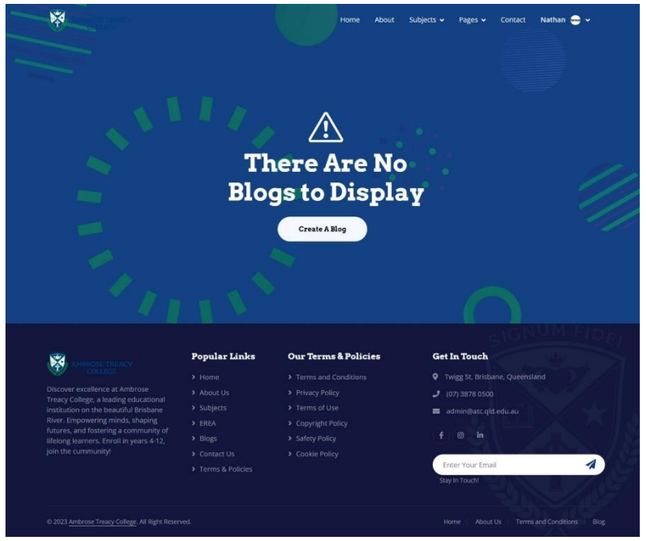

401
Unauthorized
You are not authorized to access this page. Please log in with the correct credentials and try again or return home.
Go Back To Home

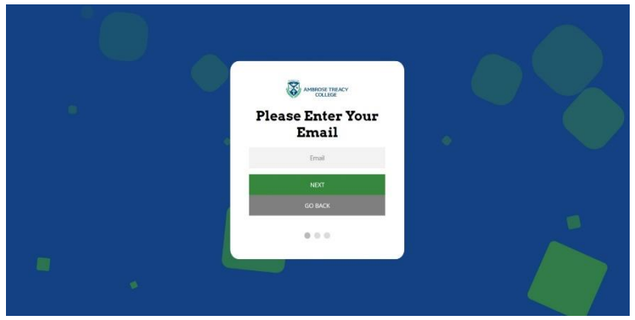

Popular Links
Home
About Us
Subjects
EREA
Blogs
Contact Us
Terms & Policies
Our Terms & Policies
Terms and Conditions
Privacy Policy
Terms of Use
Copyright Policy
Safety Policy
Cookie Policy
Get In Touch
Twigg St, Brisbane, Queensland
(07) 3878 0500
admin@atc.qld.edu.au
Enter Your Email
Stay In Touch!

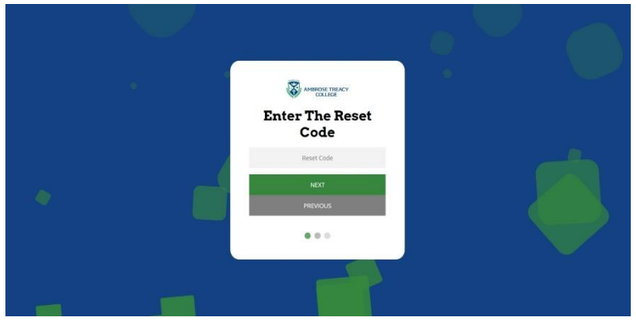

Home About Subjects Pages Contact Nathan
404 Error
Home / 404 Error

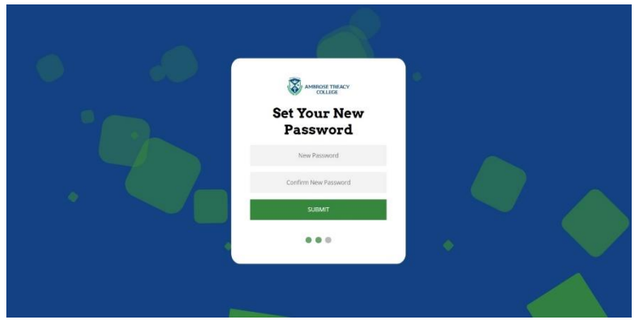

404
Page Not Found
We're sorry, the page you have looked for does not exist in our website! Maybe go back to our home page or user the navigation bar?
Go Back To Home


Popular Links
Home
About Us
Subjects
EREA
Blogs
Contact Us
Terms & Policies
Our Terms & Policies
Terms and Conditions
Privacy Policy
Terms of Use
Copyright Policy
Safety Policy
Cookie Policy
Get In Touch
Twigg St, Brisbane, Queensland
(07) 3878 0500
admin@atc.qld.edu.au
Enter Your Email
Stay In Touch!
© 2023 Ambrose Treacy College. All Right Reserved.
Home About Us Terms and Conditions Blog

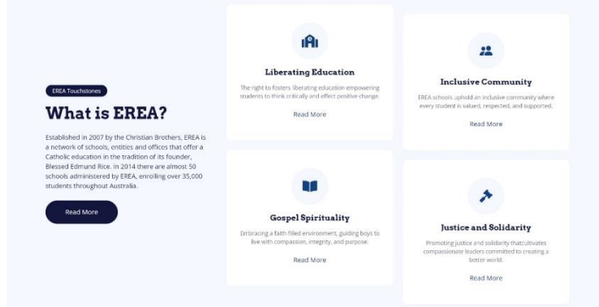

Home About Subjects Pages Contact Nathan
There Are No Blogs to Display
Create A Blog
Popular Links
Home
About Us
Subjects
EREA
Blogs
Contact Us
Terms & Policies
Our Terms & Policies
Terms and Conditions
Privacy Policy
Terms of Use
Copyright Policy
Safety Policy
Cookie Policy
Get In Touch
Twigg St, Brisbane, Queensland
(07) 3878 0500
admin@atc.qld.edu.au
Enter Your Email
Stay In Touch!
© 2023 Ambrose Treacy College. All Right Reserved.
Home About Us Terms and Conditions Blog

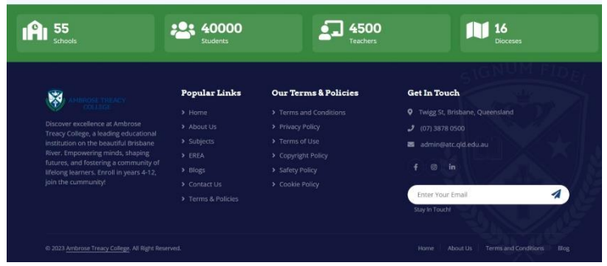

Please Enter Your Email
Email
NEXT
GO BACK

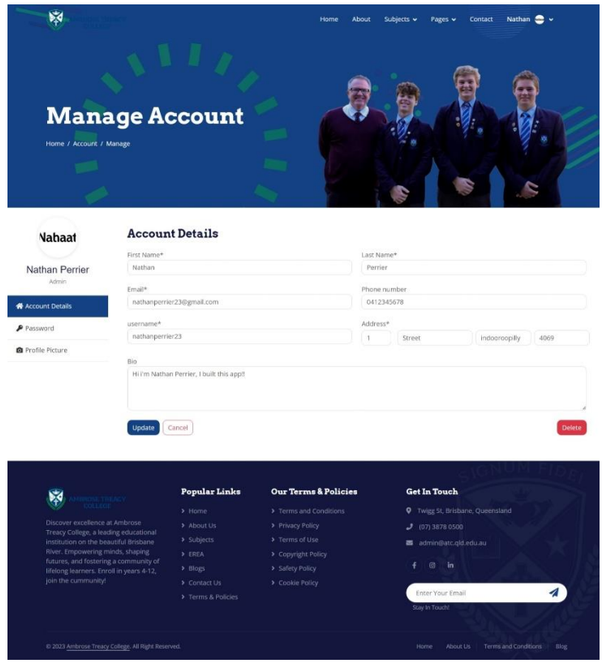

Enter The Reset Code
Reset Code
NEXT
PREVIOUS

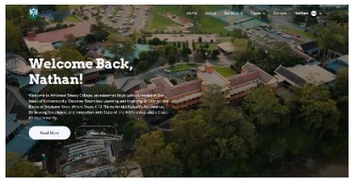

Set Your New Password
New Password
Confirm New Password
SUBMIT

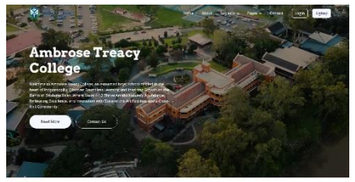

Home About Subjects Pages Contact Login Signup
Edmund Rice Education Australia
Home / EREA

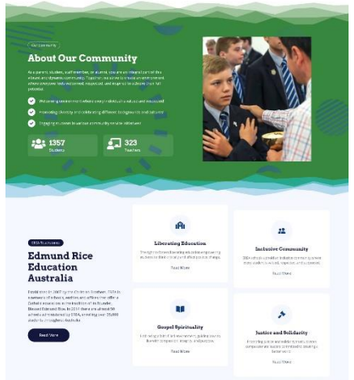

EREA Touchstones
What is EREA?
Established in 2007 by the Christian Brothers, EREA is a network of schools, entities and offices that offer a Catholic education in the tradition of its founder, Blessed Edmund Rice. In 2014 there are almost 50 schools administered by EREA, enrolling over 35,000 students throughout Australia.
Read More
Liberating Education
The right to fosters liberating education empowering students to think critically and effect positive change.
Read More
Inclusive Community
EREA schools uphold an inclusive community where every student is valued, respected, and supported.
Read More
Gospel Spirituality
Embracing a faith-filled environment, guiding boys to live with compassion, integrity, and purpose.
Read More
Justice and Solidarity
Promoting justice and solidarity thatcultivates compassionate leaders committed to creating a better world.
Read More

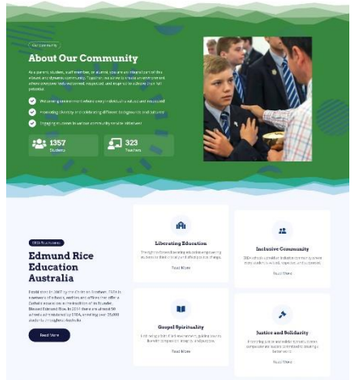

55 Schools
40000 Students
4500 Teachers
16 Dioceses
Popular Links
Home
About Us
Subjects
EREA
Blogs
Contact Us
Terms & Policies
Our Terms & Policies
Terms and Conditions
Privacy Policy
Terms of Use
Copyright Policy
Safety Policy
Cookie Policy
Get In Touch
Twigg St, Brisbane, Queensland
(07) 3878 0500
admin@atc.qld.edu.au
Enter Your Email
Stay In Touch!
© 2023 Ambrose Treacy College. All Right Reserved.
Home About Us Terms and Conditions Blog


Home About Subjects Pages Contact Nathan
Manage Account
Home / Account / Manage
Nathan Perrier
Admin
Account Details
Password
Profile Picture
Account Details
First Name*
Nathan
Last Name*
Perrier
Email*
nathanperrier23@gmail.com
Phone number
0412345678
username*
nathanperrier23
Address*
1
Street
Indooroopilly
4069
Bio
Hi I'm Nathan Perrier, I built this app!!
Update
Cancel
Delete
Popular Links
Home
About Us
Subjects
EREA
Blogs
Contact Us
Terms & Policies
Our Terms & Policies
Terms and Conditions
Privacy Policy
Terms of Use
Copyright Policy
Safety Policy
Cookie Policy
Get In Touch
Twigg St, Brisbane, Queensland
(07) 3878 0500
admin@atc.qld.edu.au
Enter Your Email
Stay In Touch!
© 2023 Ambrose Treacy College. All Right Reserved.
Home About Us Terms and Conditions Blog


Welcome Back,
Nathan!

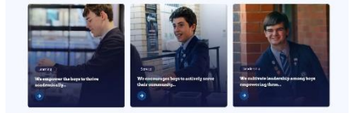

Ambrose Treacy
College

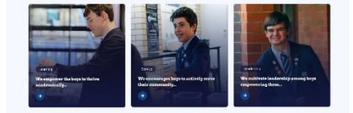

About Our Community
1357
323
Edmund Rice
Education
Australia


About Our Community
1357
323
Edmund Rice
Education
Australia

Our Campus

Our Campus


Our School Values

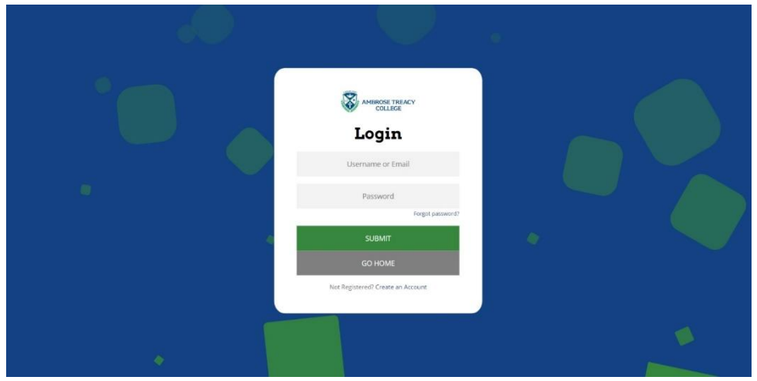

Our School Values

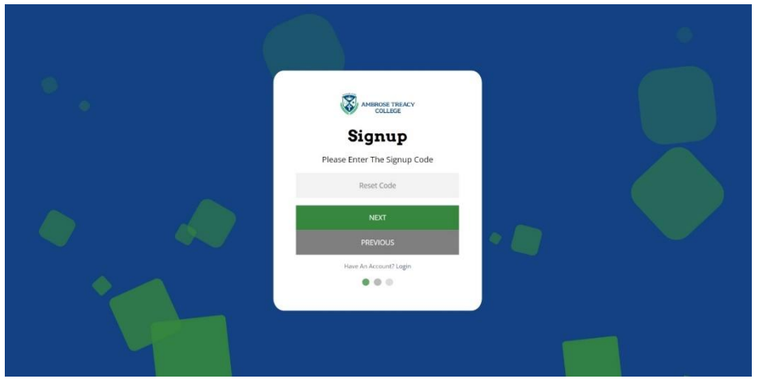

Junior
Middle
Senior
Feel Free To
Contact Us!
24/7 Support
Meet Some of Our
Friendly Staff

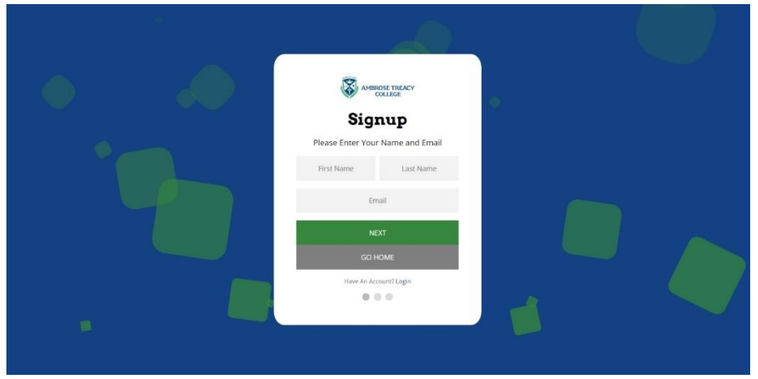

Junior
Middle
Senior
Feel Free To
Contact Us!
24/7 Support
Meet Some of Our
Friendly Staff

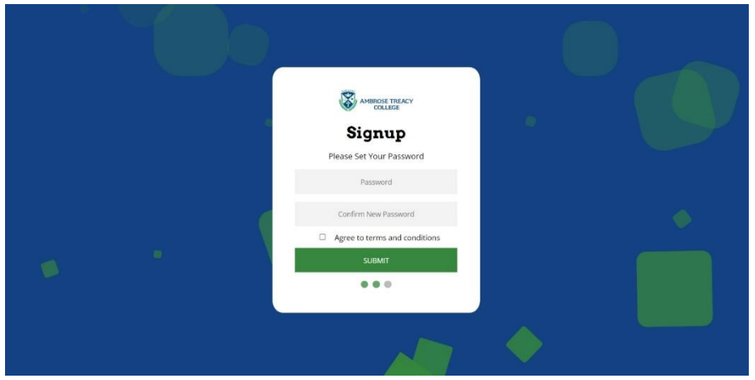

Popular Links
Our Terms & Policies
Get In Touch

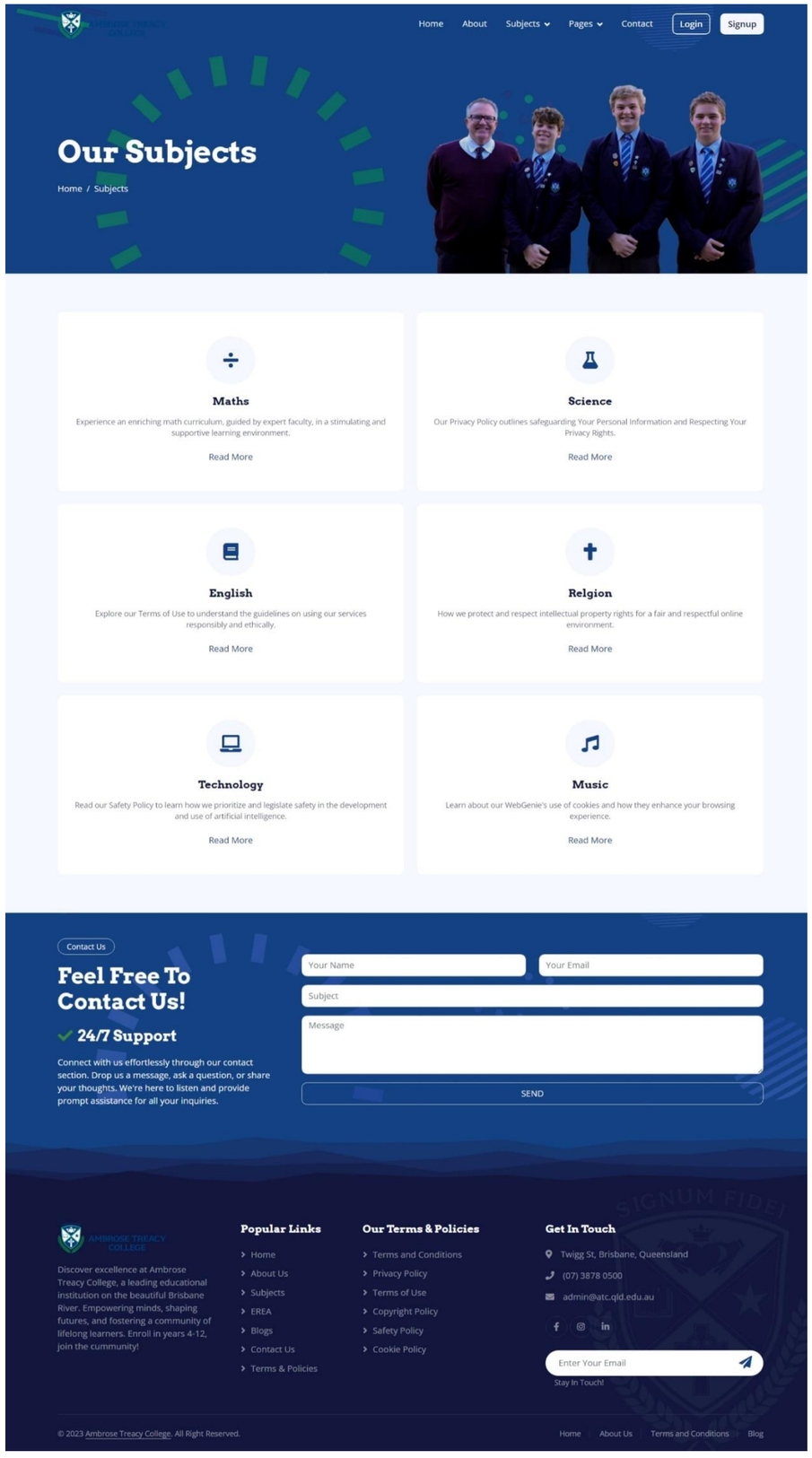

Popular Links
Our Terms & Policies
Get In Touch

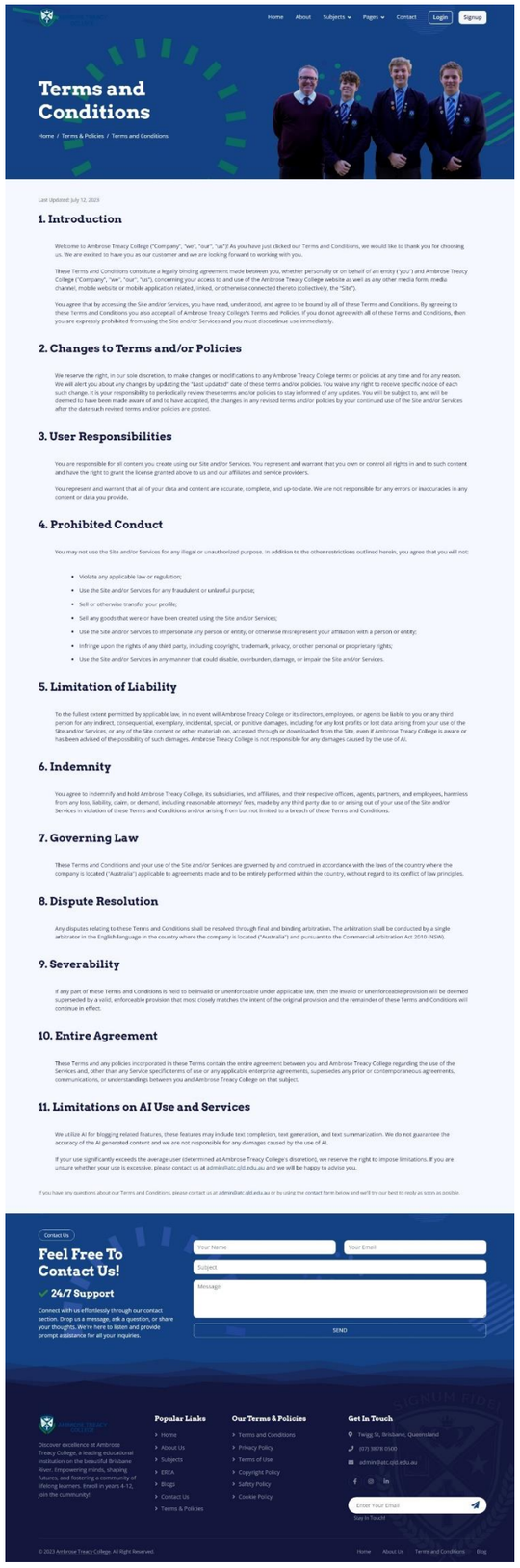

Login
Username or Email
Password
Forgot password?
SUBMIT
GO HOME
Not Registered? Create an Account

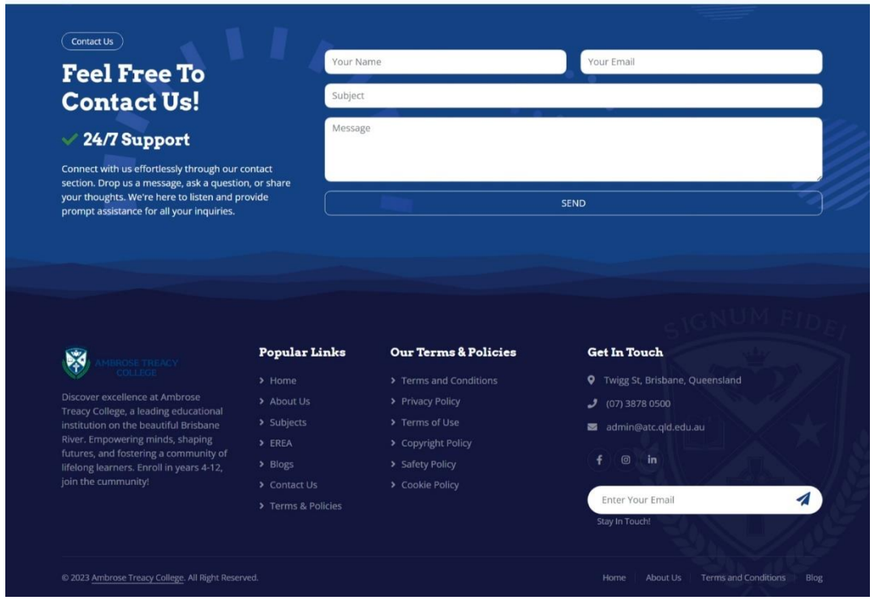

Signup
Please Enter The Signup Code
Reset Code
NEXT
PREVIOUS
Have An Account? Login


Signup
Please Enter Your Name and Email
First Name
Last Name
Email
NEXT
GO HOME
Have An Account? Login


Signup
Please Set Your Password
Password
Confirm New Password
Agree to terms and conditions
SUBMIT


Home
About
Subjects
Pages
Contact
Login
Signup
Our Subjects
Home / Subjects
Maths
Experience an enriching math curriculum, guided by expert faculty, in a stimulating and supportive learning environment.
Read More
Science
Our Privacy Policy outlines safeguarding Your Personal Information and Respecting Your Privacy Rights.
Read More
English
Explore our Terms of Use to understand the guidelines on using our services responsibly and ethically.
Read More
Relgion
How we protect and respect intellectual property rights for a fair and respectful online environment.
Read More
Technology
Read our Safety Policy to learn how we prioritize and legislate safety in the development and use of artificial intelligence.
Read More
Music
Learn about our WebGenie's use of cookies and how they enhance your browsing experience.
Read More
Contact Us
Feel Free To
Contact Us!
24/7 Support
Connect with us effortlessly through our contact section. Drop us a message, ask a question, or share your thoughts. We're here to listen and provide prompt assistance for all your inquiries.
Your Name
Your Email
Subject
Message
SEND
Discover excellence at Ambrose Treacy College, a leading educational institution on the beautiful Brisbane River. Empowering minds, shaping futures, and fostering a community of lifelong learners. Enroll in years 4-12, join the community!
Popular Links
Home
About Us
Subjects
EREA
Blogs
Contact Us
Terms & Policies
Our Terms & Policies
Terms and Conditions
Privacy Policy
Terms of Use
Copyright Policy
Safety Policy
Cookie Policy
Get In Touch
Twigg St, Brisbane, Queensland
(07) 3878 0500
admin@atc.qld.edu.au
Enter Your Email
Stay In Touch!
© 2023 Ambrose Treacy College. All Right Reserved.
Home
About Us
Terms and Conditions
Blog





Biology

In the world of biology, students explore the wonders of life itself. They study everything from the intricate workings of cells to the complexity of ecosystems. Through engaging experiments and fieldwork, students gain a profound understanding of the interconnectedness of living organisms.

Chemistry

Chemistry delves into the composition and behavior of matter. Our chemistry courses equip students with the skills to analyze substances, conduct experiments, and understand chemical reactions. It's a subject that bridges the gap between the microscopic and macroscopic world.

Physics

Physics unlocks the secrets of the physical universe. Students explore concepts like motion, energy, and the fundamental laws governing the cosmos. Our physics courses challenge students to think critically and develop problem-solving skills essential for tackling real-world challenges.