Assignment Guidance and Front Sheet

This front sheet for assignments is designed to contain the brief, the submission instructions, and the actual student submission for any WMG assignment. As a result the sheet is completed by several people over time, and is therefore split up into sections explaining who completes what information and when. Yellow highlighted text indicates examples or further explanation of what is requested, and the highlight and instructions should be removed as you populate 'your' section. This sheet is only to be used for components of assessment worth more than 3 CATS (e.g. for a 15 credit module, weighted more than 20%; or for a 10 credit module, weighted more than 30%).

To be completed by the student(s) prior to final submission:
Your actual submission should be written at the end of this cover sheet file, or attached with the cover sheet at the front if drafted in a separate file, program or application.

Student ID	**1921983**

To be completed (highlighted parts only) by the programme administration after approval and prior to issuing of the assessment; to be consulted by the student(s) so that you know how and when to submit:

Date set	22rd November 2021
Submission date (excluding extensions)	07th February 2022 12 noon (mid-day)
Submission guidance	**Submission requirements** • You must submit a **project report indicating the Student ID number** in the title of the submission, i.e., ***0000001_Report.pdf***. • The report must be in **PDF** format. • The report must be submitted via Tabula and must not be Zipped. • You must zip all the codes used for your implementation README file, and submit the zipped file **indicating the Student ID number** in the title of the submission, i.e., ***0000001_Code.zip***. • You **must** check if the report and the zipped file have been uploaded successfully. • You must include the assessment front sheet in your report. **Report Requirements** • The report should include a title page, table of contents in the report. • The report should include your student ID. • There is no page limit as long as it fits the total number of words for the report. • The report should follow a logical and well-defined structure with headings and subheadings. • Diagrams in the report should be clearly labelled and well-presented. • Appendices will not normally be marked but they must not include material essential to the argument developed in the main body of the work.

Marks return date (excluding extensions)	Within 20 working days after the submission deadline.
Late submission policy	If work is submitted late, penalties will be applied at the rate of **5 marks per University working day** after the due date, up to a **maximum of 10 working days** late. After this period the mark for the work will be reduced to 0 (which is the maximum penalty). "Late" means **after the submission deadline time as well as the date** – work submitted after the given time even on the same day is counted as 1 day late.
Resubmission policy	If you fail this assignment or module, please be aware that the University allows students to remedy such failure (within certain limits). Decisions to authorise such resubmissions are made by Exam Boards. Normally these will be issued at specific times of the year, depending on your programme of study. More information can be found from your programme office if you are concerned.

To be <u>completed</u> by the <u>module owner/tutor</u> prior to approval and issuing of the assessment; to be <u>consulted</u> by the <u>student(s)</u> so that you understand the assignment brief, its context within the module, and any specific criteria and advice from the tutor:

Module title & code	WM393 Software Development Life Cycle
Module owner	Dr. Jianhua Yang
Module tutor	Dr. Jianhua Yang & Dr. Young Saeng Park
Assessment type	Written report & Implementation
Weighting of mark	60% of the total module mark

Assessment brief
Scenario Refer to the scenario in the assignment 1. **Task** Based on the function you selected in the assignment 1, the assignment 2 extends on the assignment 1 and additionally performs the following two tasks. First, you have to produce an individual report for the function (board) that you selected in the assignment 1. You should include the contents from your assignment 1 in the individual report but needs to improve the contents for further requirements and implementation. The detail instructions for the individual report are described below. Second, you have to implement partial or close to a workable version of software based on the requirements and architecture design. Remember that it is not only the design of user interface, but you should implement some required features for the function you selected. The language decision is up to you, but the implementation should be based on the requirements defined in your individual report you have created. The detail instructions for the implementation are given below.

[Individual Report]

Here are the instructions of how to produce your individual report:

- You should include the contents for your function defined in the assignment 1 such as the functional requirements, non-functional requirement, user interface, etc.
- If necessary, you are free to improve the contents for your function previously defined in the assignment 1.
- If necessary, you are free to include some other contents related to your function from the assignment 1 such as the management of functions control, the management of users, non-functional requirements, etc.
- To improve the structural view in the architecture design, you must include at least one class diagram for your function.
- To improve the behavioural view in the architecture design, you must include at least one sequence diagram for your function.
- To include implementation technical review, you must include some important parts of your code and technical description with some screen captures. Remember that the code is not included in the total word counts but the description is included. So, put the code inside a table with a caption in order to make it an easy to recognise.
- To verify your implementation code, you must include at least one test case for some functions with testing code and description. Remember that the testing code is not included in the total word counts but the description is included. So, put the testing code inside a table with a caption in order to make it an easy to recognise.
- You must include the progression of your work. It is about how effectively you evolve your work considering agility, milestones, etc.
- You must include your current status and future work. The current status and future work are about what you have completed based on the functional, non-functional and user interface requirements, and what you need to improve in future.
- You must include a short conclusion.
- You can add more appendixes if necessary. Remember it is not included in the total word counts.
- This is only a proposed report format, but you can change the format if necessary.
    - Cover page
    - Table of Contents
    - Introduction
    - Overall description
    - Requirements
    - Design
    - Technical Detail
    - Test Design
    - Progression of Work
    - Current Status and Future Work
    - Conclusion
    - Appendixes

[Implementation]

Here are the instructions of how you implement your software and how you submit your code:

- The implementation must be based on the requirements defined in your report.
- You should include comments in your code to provide better understanding and analysis of your code for readers.

- You should include a README file which includes how to setup and how to execute your software and additional information required for your software. There is no format restriction in the README file such as text only, markdown, json, xml, etc. As long as it is understandable by readers, it will be acceptable.
- However, if any problems do occur due to not providing the detail and clear explanations of setup and execution of your software in a README file, it is your responsibility for the problems.
- In a case that your software is available online, the README file should include how to access your software.
- You should provide a zip file containing all source code but should not include dependencies. Instead, you should provide the instructions in the README file explaining how to install them if you need dependencies for your software.
- In a case that it is difficult to execute or access your software, you should include as many screen-captured images as possible in your report or provide the images separately with your code in a zip file.
- You must include some test codes in the zip file and also include some description regarding these tests in your report.
- It is important that your code has to be well-arranged. If it is necessary, please consider refactoring your code before submitting it.

Word count	2500 words ± 10%. Word count is defined as the number of words contained within the main body of the text which include titles, headings, abstracts, summaries, in-text citations, quotations, and footnotes. Items excluded from the word count are acknowledgements, tables of contents, a list of acronyms, meeting notes, a glossary, a list of tables, or figures. Exceeding the work count: For more than 10% up to and including 20% a deduction of 10 percentage points will be applied. For more than 20% up to and including 30% a deduction of 15 percentage points will be applied. More than 30%, The work will be assigned a grade of 0.
Module learning outcomes (numbered)	1. Demonstrate a sound understanding of a range of software process models and symbolic representations. 2. Discriminate scenarios where different design patterns and software testing strategies can be applied, and critically evaluate these patterns and testing solutions. 3. Apply the range of software tools used tools for configuration management, version control and software build. 4. Discriminate the key concepts and techniques used in the Agile Manifesto and Scrum, carefully design and critically evaluate project plans using these techniques. 5. Distinguish current and emerging XP, Lean, and Kanban values, principles, and practices, use these techniques to analyse and optimize process workflow.
Learning outcomes assessed in this	LO3, LO4, LO5

assessment (numbered)	
Marking guidelines	First class report is expected to be exceptional works by demonstrating excellent analysis of the project scenario and task, excellent specification of functional and non-functional specification, excellent design of system architecture, and excellent partial or close to a workable version of software. Second class report is expected to be works by demonstrating good analysis of the project scenario and task, good specification of functional and non-functional specification and excellent design of system architecture, and good partial or close to a workable version of software. Report that presents a limited quality work by demonstrating some relevant knowledge with partially evident critical thinking will be deemed as third class. Work that is below the standard required for the appropriate stage of an Honours degree will be deemed as fail. ** Detailed marking rubrics can be found in the assignment brief.
Academic guidance resources	**How to seek further help** Students are strongly advised to ask tutors via Moodle forum • https://warwick.ac.uk/services/library/students/your-library-online/ Numerous online courses provided by the University library to help in academic referencing, writing, avoiding plagiarism and a number of other useful resources. **Referencing** Follow the University of Warwick referencing guidelines, found via the links: • https://warwick.ac.uk/services/library/students/referencing/referencing-styles • https://warwick.ac.uk/fac/soc/al-archive/leap/writing/referencing/intext/ Should you experience difficulties likely to seriously impact your ability to complete any module work, please see the website section for Mitigating Circumstances and Reasonable Adjustments at: • https://warwick.ac.uk/services/aro/dar/quality/categories/examinations/policies/u_mitigatingcircumstances/

WM393 Assignment 2 (WMGTSS Dev)

Table of Contents

Introduction

WMG TSS shall be used to increase Tutor-Student interaction by encouraging online discussion and file sharing, specifically due to COVID-19. This report details all stages of the development lifecycle, from concept creation all the way to testing and deployment.

The project shall be delivered using the following languages and frameworks:

- Python 3.8
  - Flask
  - SQLAlchemy
  - BCrypt
  - PyTest
- HTML – Bootstrap
- CSS
- Docker
- Heroku

Problem Statement

> *The pandemic has highlighted inadequacies within the existing resource and lecture management systems in place at WMG. These systems are creating unnecessary complications for students, resulting in an inconsistent user experience. Students and tutors are spending an unacceptable quantity of time identifying the best place to upload and access resources.*

Overall description

I chose the *Flask* framework due to its versatile nature and that it lends itself to quick and manageable web-development. *Flask* provides tools, libraries, and mechanics that allow you to build a web app with no dependencies and won't force you to design a certain way. *Flask* has integrated SQL database functionality and user management, which has allowed for rapid development of the application. All my *Flask* templates are using *HTML5* (with the *Bootstrap* framework) and *CSS*. To package prototype web-apps I plan to use *Docker* and to deploy production apps I shall use *Heroku* to host the web-app.

Development Lifecycle

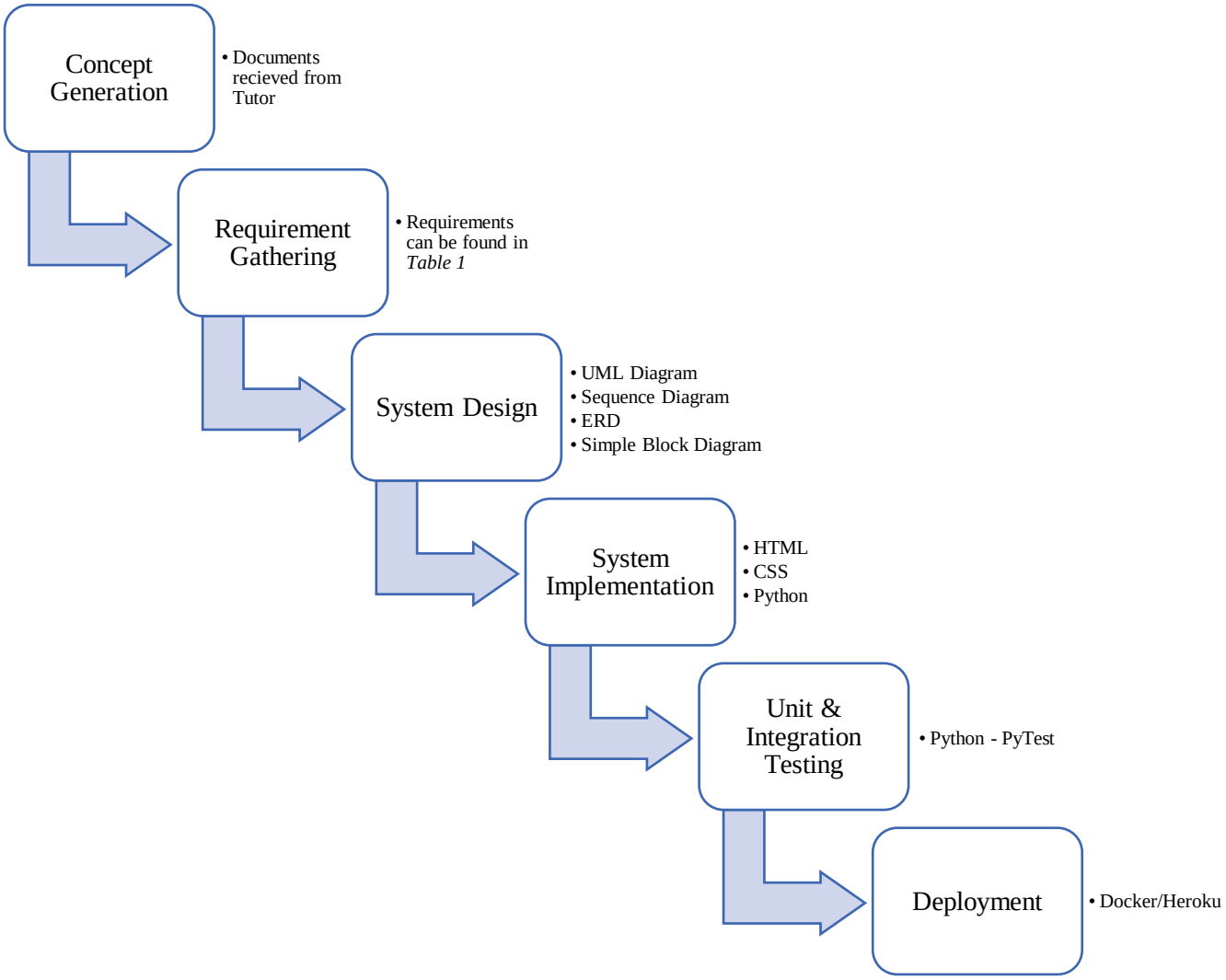

Figure 1: SDLC Workflow

Progression of Work

GitHub: *https://github.com/jaxkyoung/1921983_WM393_Assignment_2.git*

For this project, I have used *git* and *GitHub* for version control following a standard workflow. Once the MVP (Minimum Viable Product) is developed, I will implement newly requested features using branches (See *figure 2*). Each feature will have its own branch and will be tested independently (unit tested) before it is merged (once an integration test is passed). Each version deployed to production can be tagged in my repository with a version number (See *figure 3*).

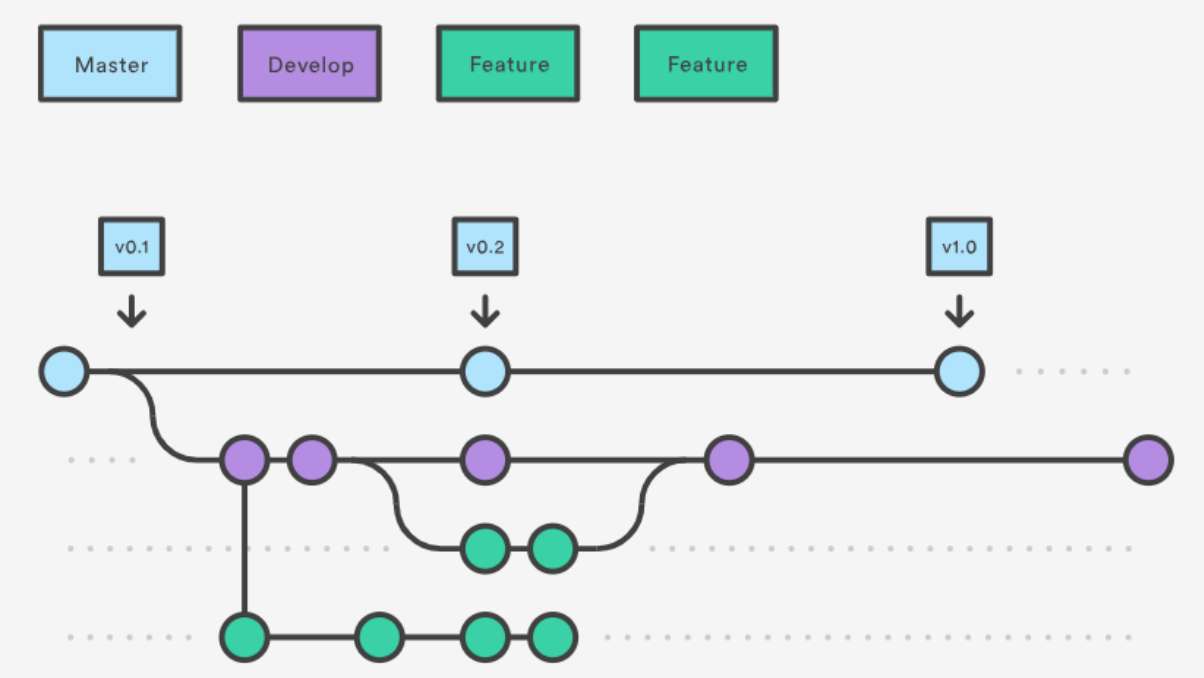

Figure 2: git Workflow

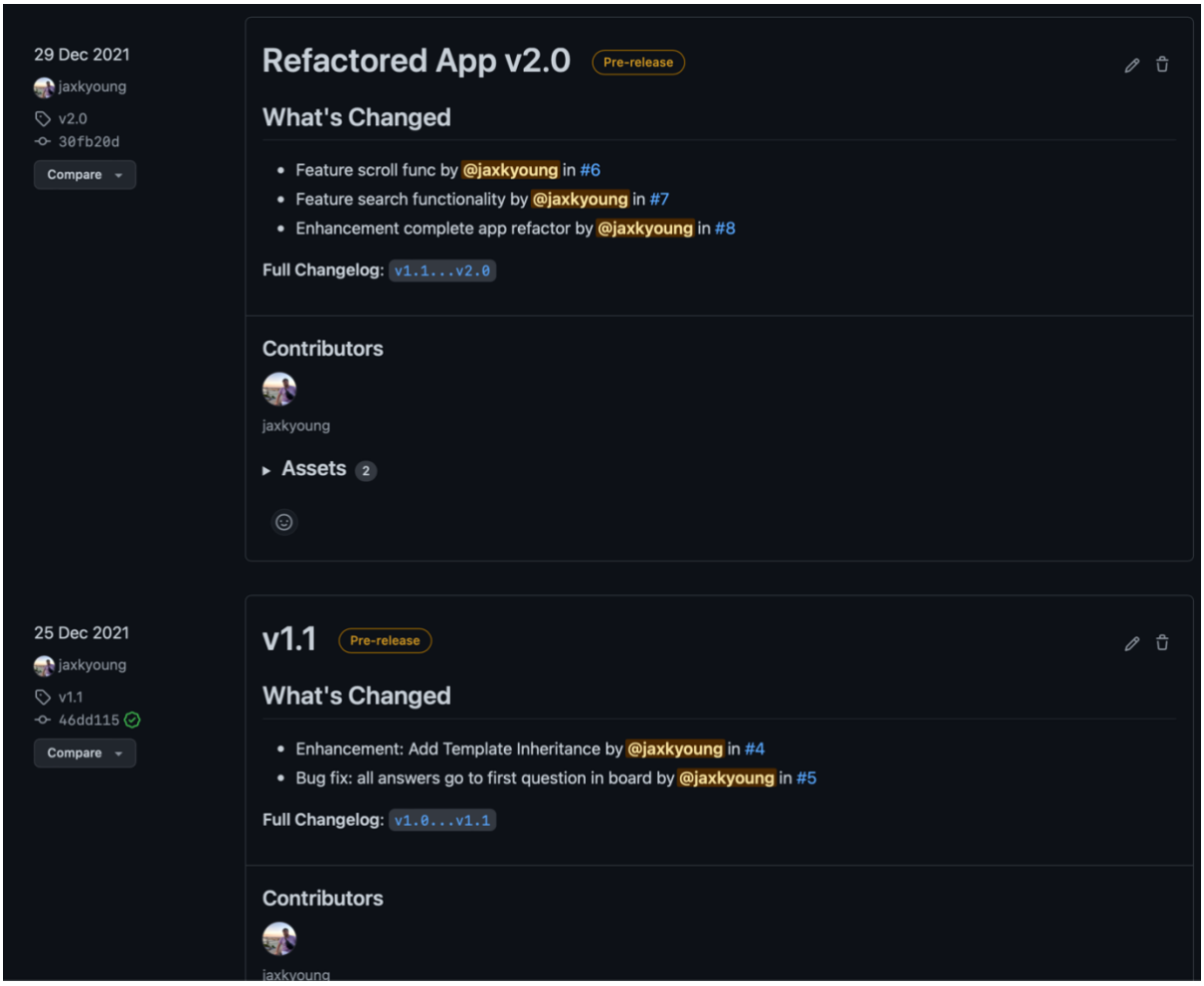

Figure 3: git Versions and Tags

Requirements

Table 1: Full Collection of Requirements

Requirement ID	Requirement
User Management	
UM_F_1	Users shall be able to create an account of three types (Tutor, Teaching assistant, Student)
UM_F_2	Users must include first name, last name, email, and password
UM_F_3	All passwords shall be hashed before storage in the database
UM_F_4	All tutor accounts created must be approved by another tutor account before they are available for use
UM_F_5	Users can be deleted upon request to the database administrator
UM_F_6	Users can be edited upon request to the database administrator
UM_F_7	Users shall register via an html form
UM_F_8	Users shall log-in via an html form
User Story: *As a tutor, I want to be able to create a Q&A board so that students can post questions.*	
Acceptance Criteria: *Given the user is a Module Lead, when the user presses create board, a board shall be created with the input title, so that students will be able to post questions on.*	
Q&A_F_1	When the user presses the create button, and the user is a tutor, a pop-up window shall be opened
Q&A _F_2	The user must include a title to the board for it to be created
Q&A _F_3	When the user presses create on the pop-up window, the board shall be created
Q&A _F_4	If the user attempts to create a board without a question, an error message shall appear
Q&A _F_5	The user may include a description or body to the board to describe its function
Q&A _NF_1	The title shall have a character limit of 100 characters
Q&A _NF_2	The system shall allow multiple users to create a board at one time and shall schedule these activities in a way that no interruption is shown to any user
Q&A _NF_3	All boards shall be backed up to the cloud database
User Story: *As a student, I want to be able to comment on a question posted on the board, so that I can discuss with other students.*	
Acceptance Criteria: *Given the user has access to a board and its questions, when the user presses the comment button, then the system shall open a commenting window.*	
Q&A _F_6	When the student posts a comment, the student shall be able to include a body of text in their answer
Q&A _F_7	There shall be no maximum number of answer posts on a question thread
Q&A _F_8	When the user presses the comment button, and the user is a student, then the user shall be able to post a comment to a question using the pop-up window
Q&A _NF_4	The description shall have a character limit of 500 characters
Q&A _I_1	The create button shall be blue with black text

Q&A _I_2	The pop-up window shall have a title box and body box
Q&A _I_3	The post button shall be blue with black text
Q&A _I_4	The discard button shall be red with black text
User Story: *As a tutor, I want to be able to post a question on the board so that I can post frequently asked questions.*	
Acceptance Criteria: *Given the user is a tutor, when they press the post button, a window shall be opened so they can fill in a form and ask a question*	
Q&A _F_10	When the user presses the create button, and the user is a tutor, then the user shall be able to post a question using the pop-up window
Q&A _F_11	When a tutor asks a question, the tutor must include a title/question in their post
Q&A _F_12	When a tutor asks a question, the tutor shall be able to attach images or documents as supporting information to their post
Q&A _F_13	The tutor shall be able to delete all posts
Q&A _NF_5	The title shall have a character limit of 500 characters
Q&A _NF_6	The body shall have a character limit of 1500 characters
User Story: *As a postgraduate assistant, I want to be able to answer questions as if I were a tutor, so that questions I know the answer to can be resolved quickly.*	
Acceptance Criteria: *Given the user is a teaching assistant, when they press the answer button, a window shall be opened so they can fill in a form and answer a question*	
Q&A _F_14	When the GTA posts an answer, the GTA shall be able to include a title in their answer
Q&A _F_15	When the GTA posts an answer, the GTA shall be able to attach images or documents as supporting information to their answer
Q&A _F_16	There shall be no maximum number of answer posts on a question thread
Q&A _F_17	When the user presses the answer button, and the GTA is a tutor, then the user shall be able to post an answer to a question using the pop-up window
User Story: *As a tutor, I want to be able to post an answer to any question on the Q&A board so that all students can see my answer.*	
Acceptance Criteria: *Given the user is a tutor, when they press the answer button, a window shall be opened so they can fill in a form and answer a question.*	
Q&A _F_20	There shall be no maximum number of answer posts on a question thread
Q&A _F_21	When the user presses the answer button, and the user is a tutor, then the user shall be able to post an answer to a question using the pop-up window
User Story: *As a student, I want to be able to post a question on the Q&A board so that the tutor can answer it.*	
Acceptance Criteria: *Given the user is a student, when they press the post button, a window shall be opened so they can fill in a form and ask a question*	

Q&A _F_22	When a student asks a question, the student shall be able to include a body of text in their post
Q&A _F_23	When the user presses the create button, and the user is a student, then the user shall be able to post a question using the pop-up window
Q&A _NF_9	The system shall have a 1-minute grace period before publicising the question
Q&A _NF_10	The file size limit shall be 50Mb

Design

UI Design

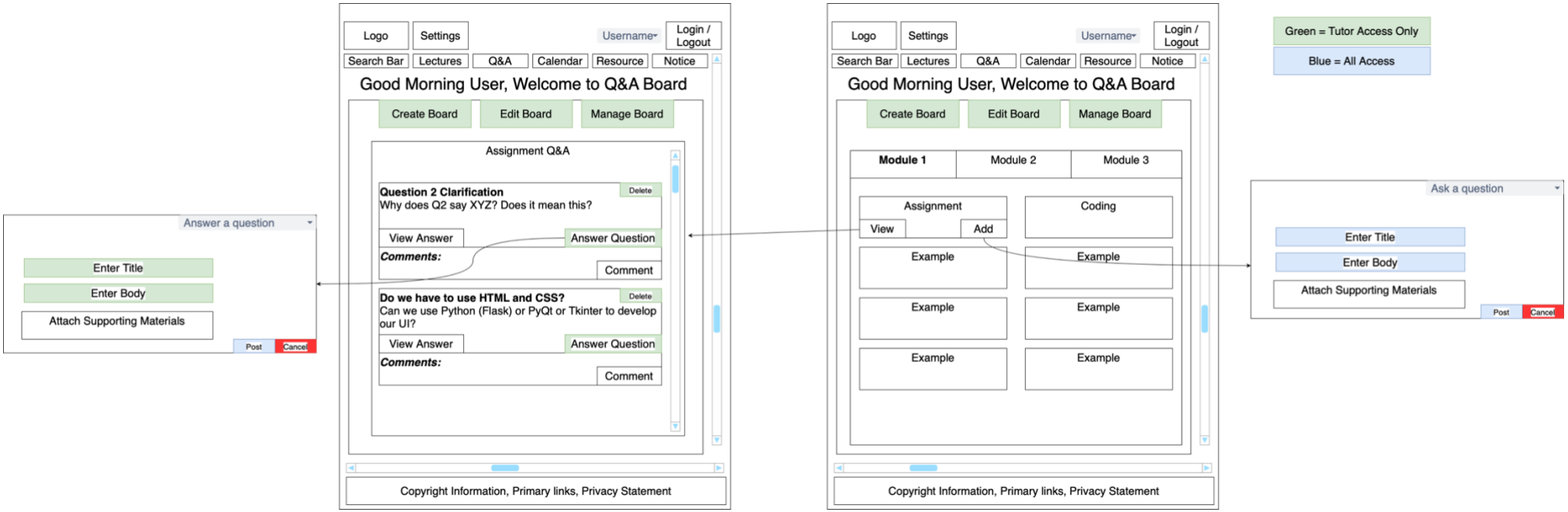

Figure 4: Final UI Design from Assignment 1

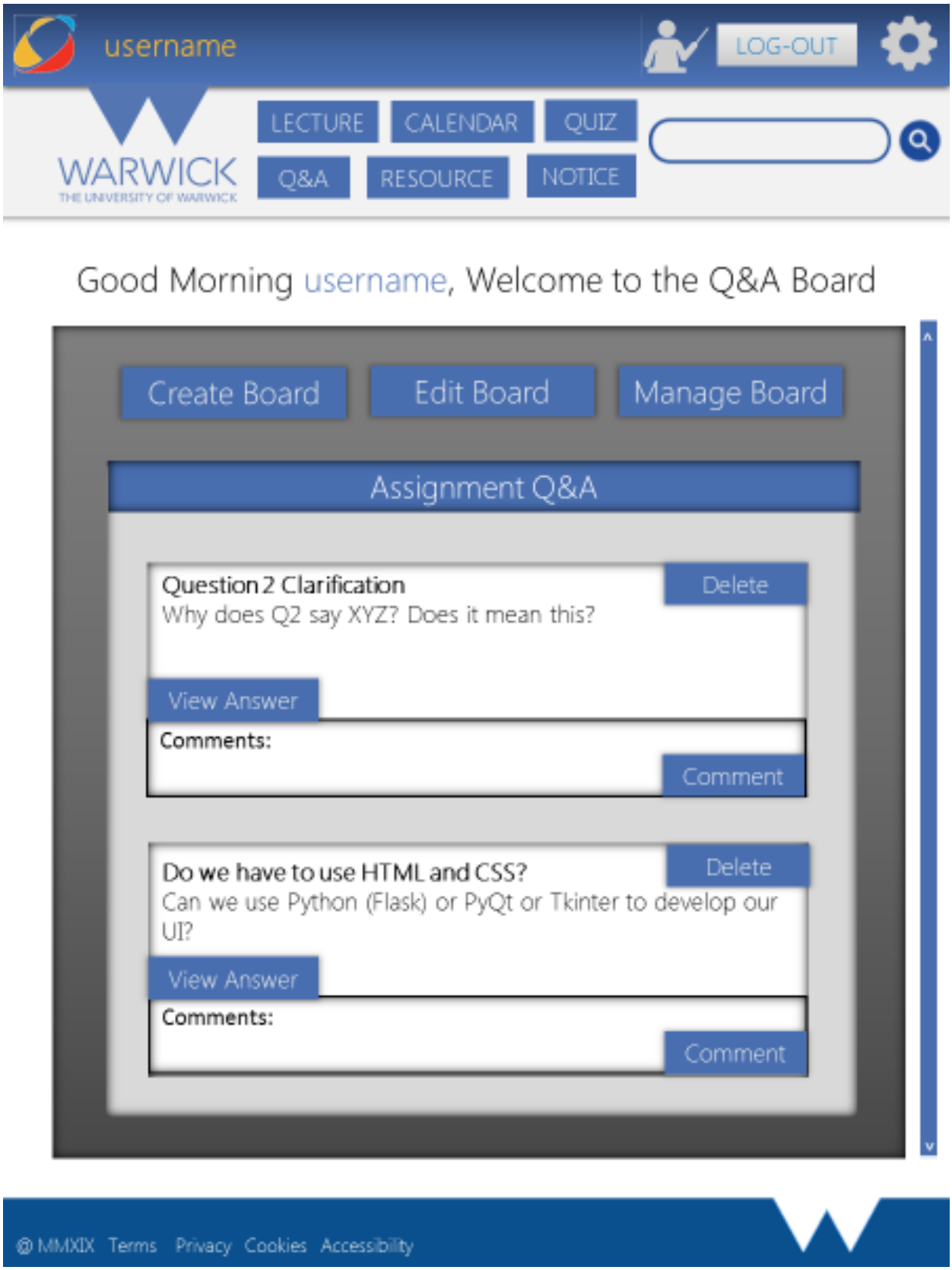

Figure 5: Enhanced UI Design

Use Case Diagram

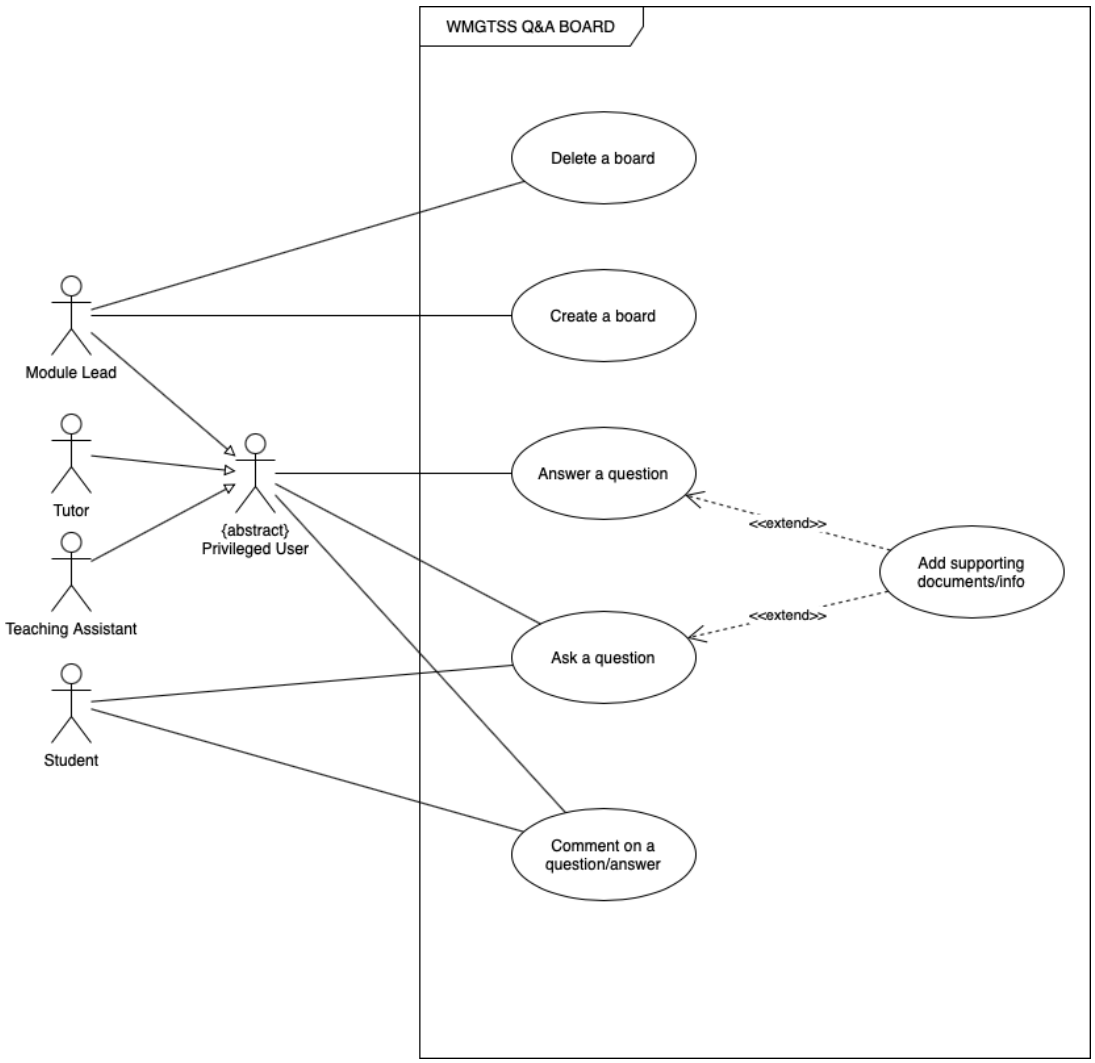

Figure 6: Main Use Case Diagram

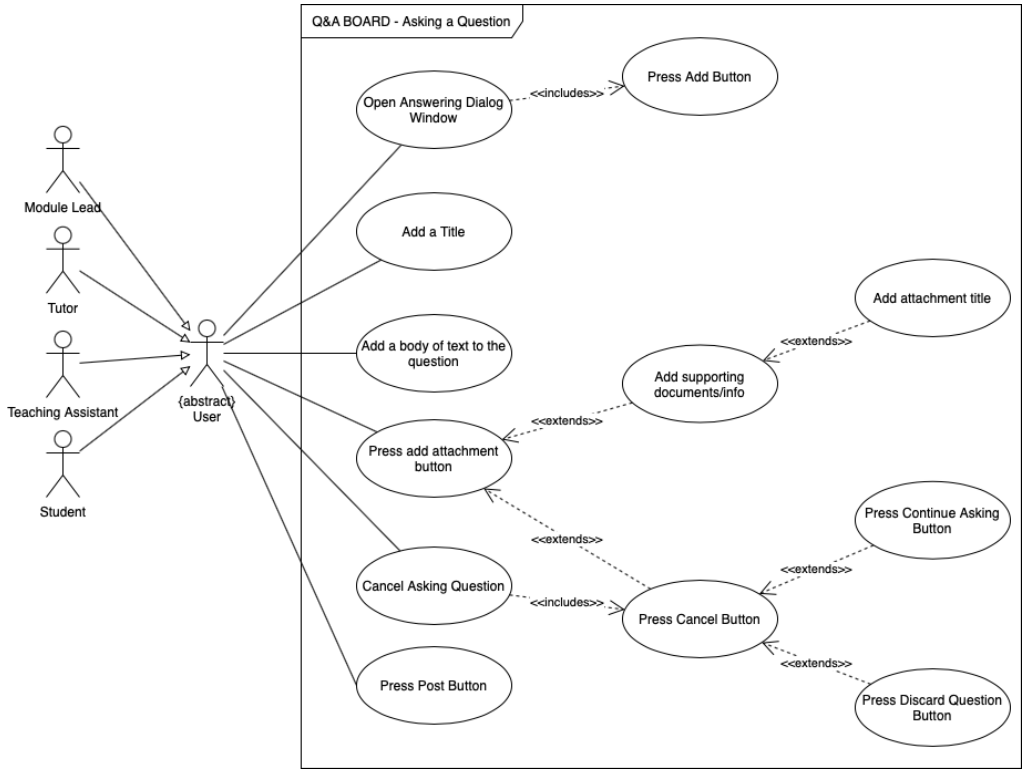

Figure 7: Specific Use Case

Use Case Thumbnail	Q&A Board - Asking a Question
Actors	Module Lead, Tutor, Teaching Assistant, Student
Use Case Description	This use case diagram describes the functionality of asking a question on the Q&A board
Precondition	• Student is registered with account on WMG TSS • Privileged User is registered with an account on WMG TSS • A Q&A board is already created
Postcondition	• Question has been asked • Tutor able to answer question • Question is displayed on relevant Q&A board
Basic Text	1. User presses add button 2. User adds a title and text to the question 3. User presses post
Alternative Flow	After step 1, the user can choose at any point to attach supporting documents
Priority	High

Simple Block Diagram

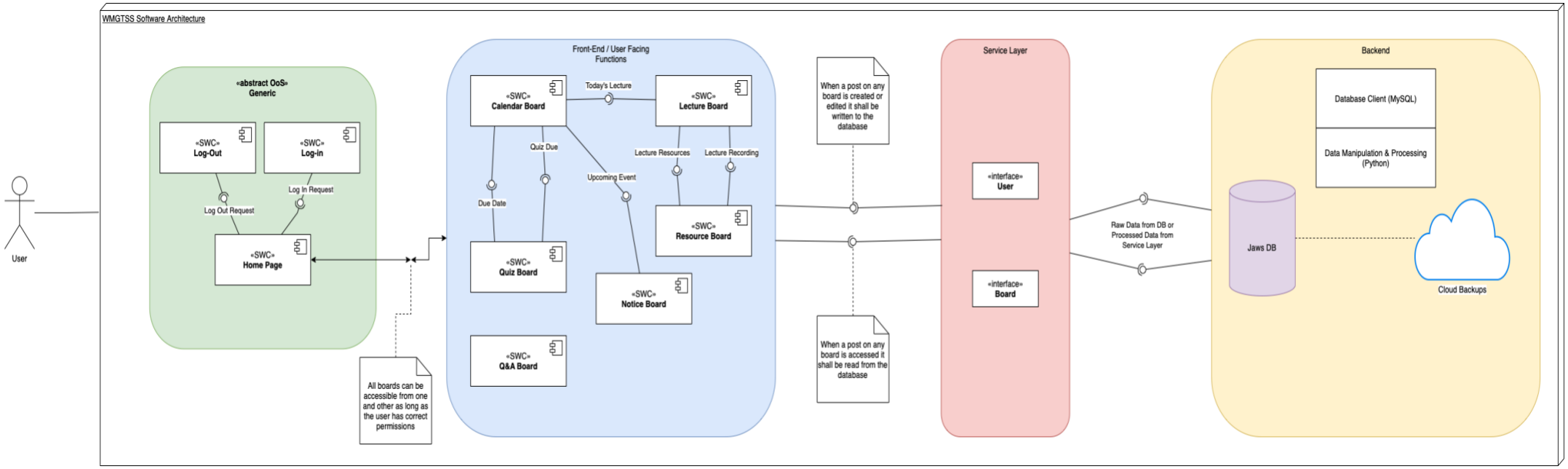

Figure 8: System Architecture

The system architecture was designed with maintenance and modularity in mind (Generic Access Functions, User-Facing Board Functions, Service Layer – read and write to backend, and the backend). Following a standard design of *Application -> Service Layer -> Backend.* Within the application layer the board is split into its own software component (SWC) allowing for complete isolation from other components. Each SWC will have defined signals being transmitted to and from itself. The service layer provides an abstract connection form front-end to backend allowing for expansion of both front and backend services. Our backend service shall initially be hosted on a cloud service with additional backups but shall have the potential for on-site physical hosting.

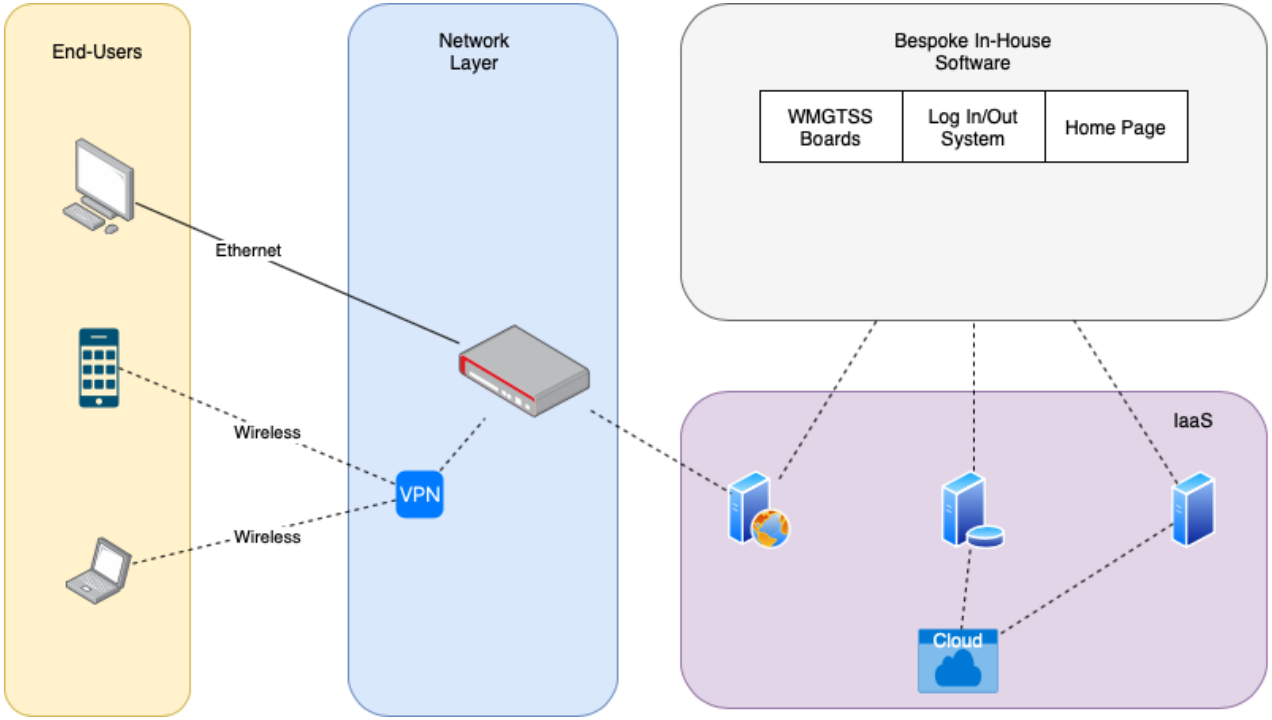

Figure 9: Physical Architecture

In the group assignment, I planned on using an IaaS platform, I have partially adopted this method, using Heroku for my webserver and using an online hosted database.

Entity Relationship Diagram

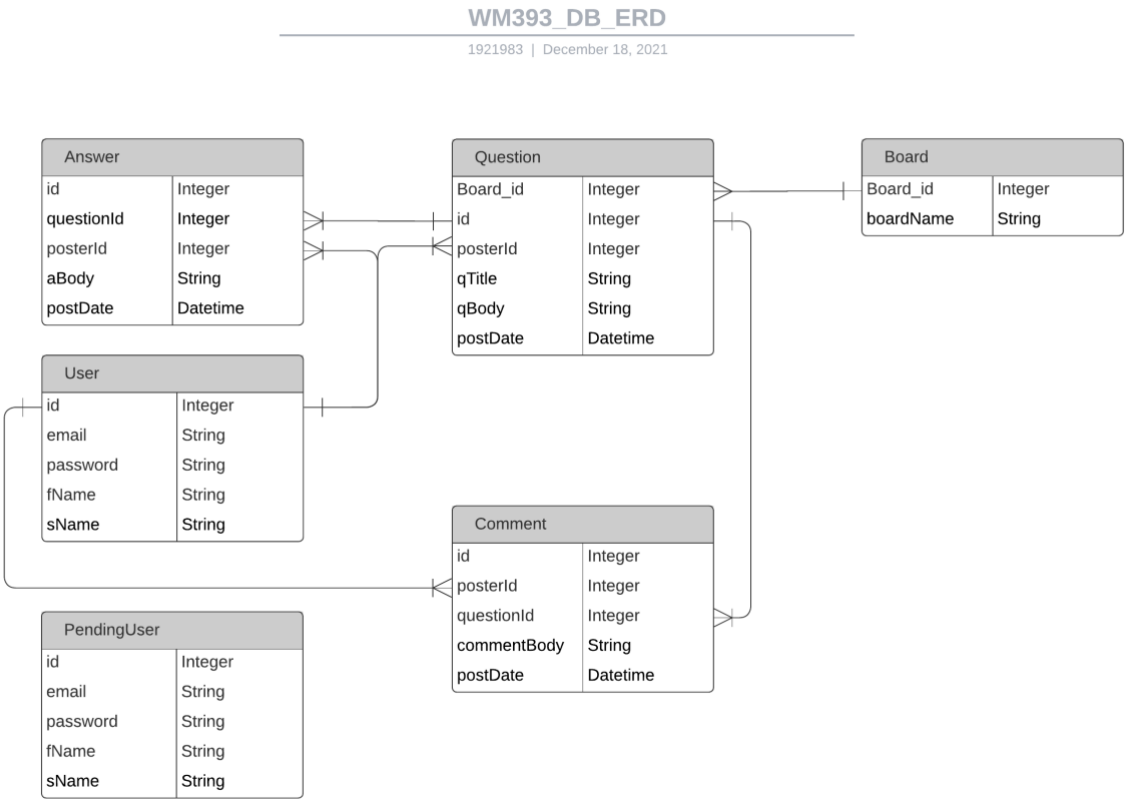

Figure 10: Initial ERD

The above diagram shows a logical representation of my database, containing user, board, question, answer, and comment tables. Each table containing a primary key, with some containing foreign keys to enforce third-normal-form. Where a database is in third normal form if there is no transitive dependency for non-prime attributes (and is also in the second normal form.) Third normal form then enforces referential integrity. Each relationship is either a 1-1, 1-Many or Many-1. There are no Many-Many relationships. This allows for data integrity and adherence to ACID rules.

UML Class Diagram

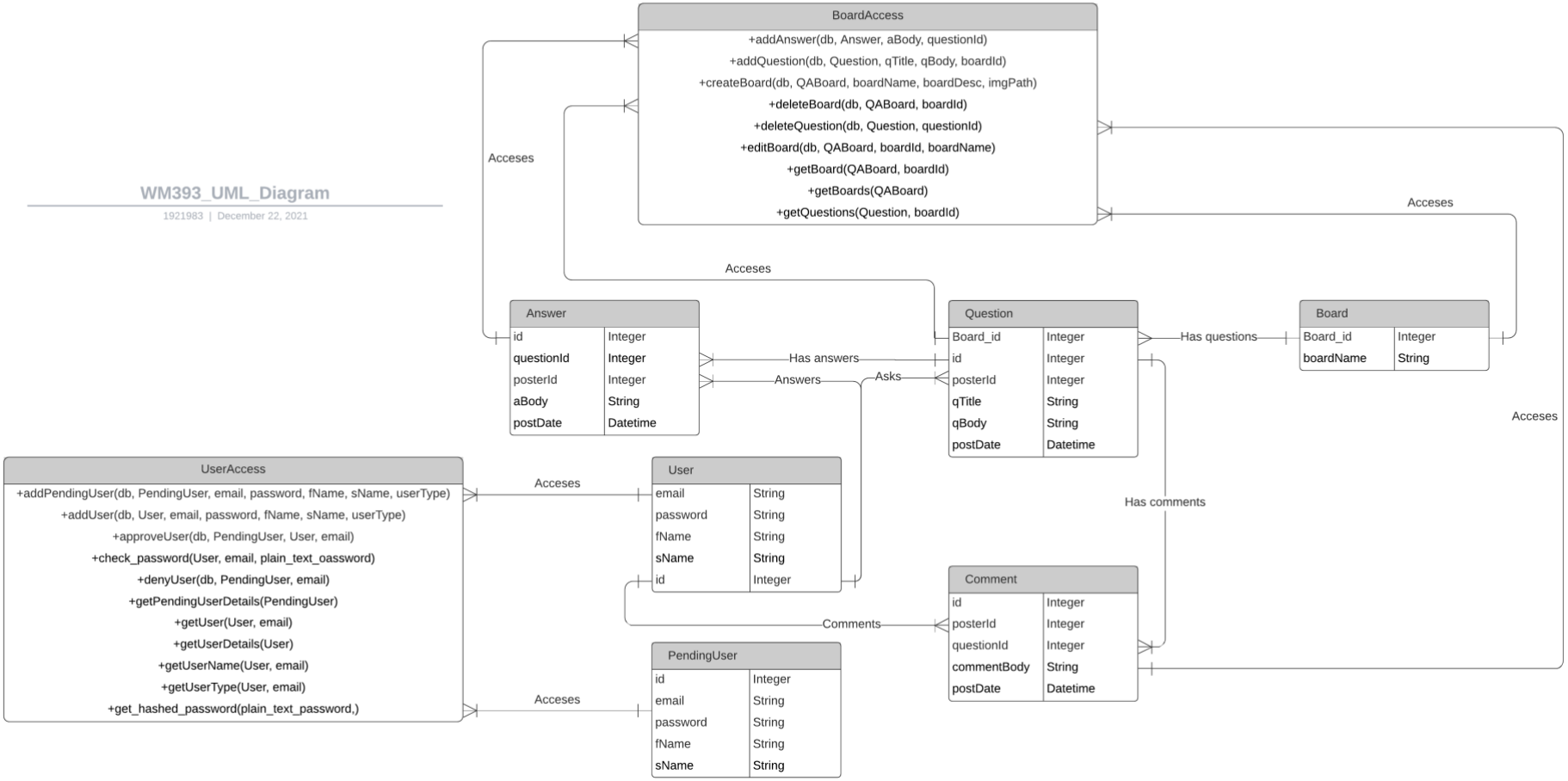

Figure 11: UML Diagram

Sequence Diagram

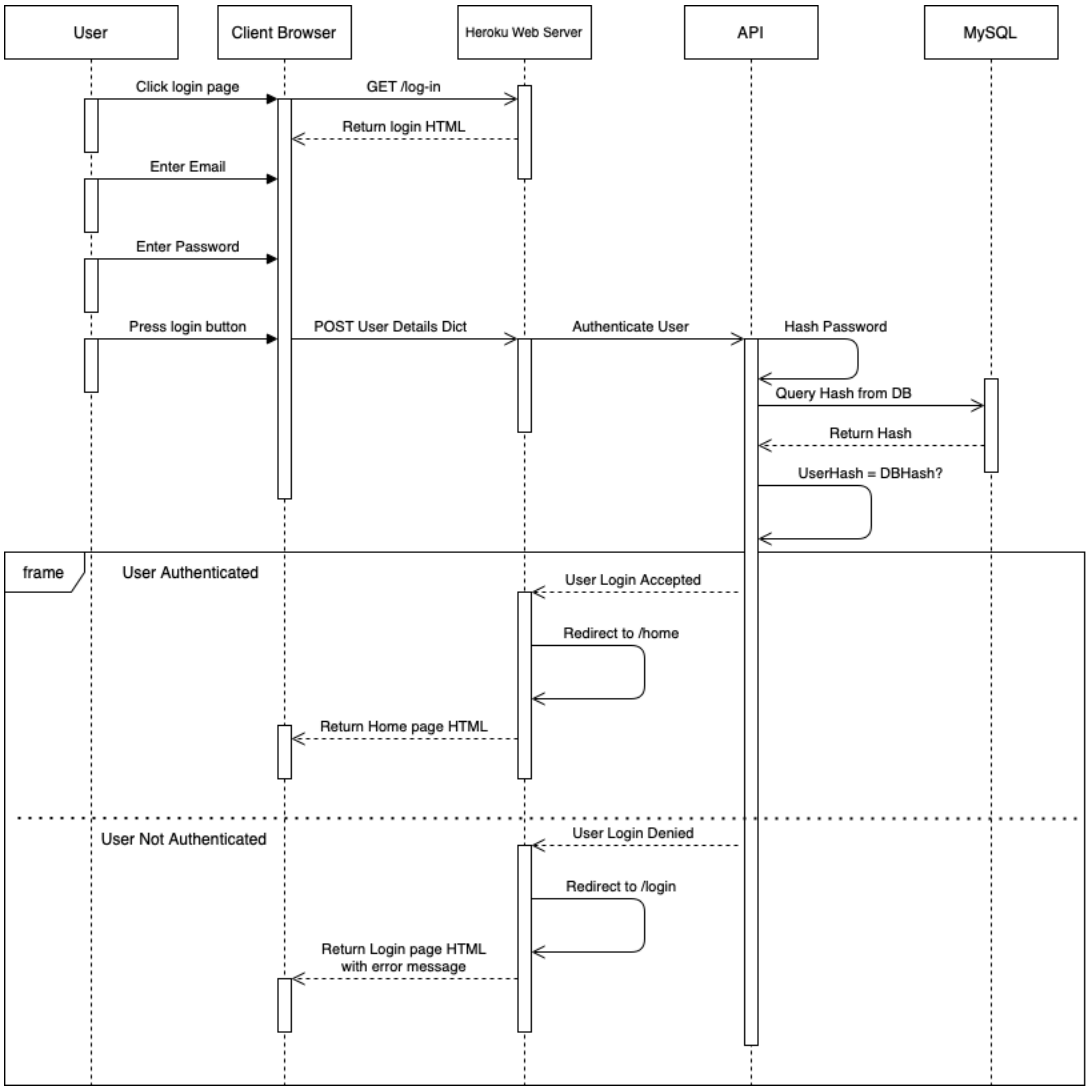

Figure 12: Authentication Sequence Diagram

Figure 7 shows the basic flow of user authentication through the *Flask* application, from the user interaction to client browser to web server and finally the database. The user can input information and press buttons on the browser, a button press on the browser executes either a GET or POST request. Usually to render a page, a GET is executed and to transmit data a POST is used. So, in this case, the user inputs their details, presses login and a POST request is executed transmitting the login details. Upon the POST request, the web server attempts to authenticate the user by hashing the input password and comparing with the password associated with the email that is stored in the database. If the passwords match, then the login is accepted, and the user is redirected to the home page. If the passwords do not match, the login is denied, and the user is reloaded to the login page with an error message.

Technical Detail

Project Structure

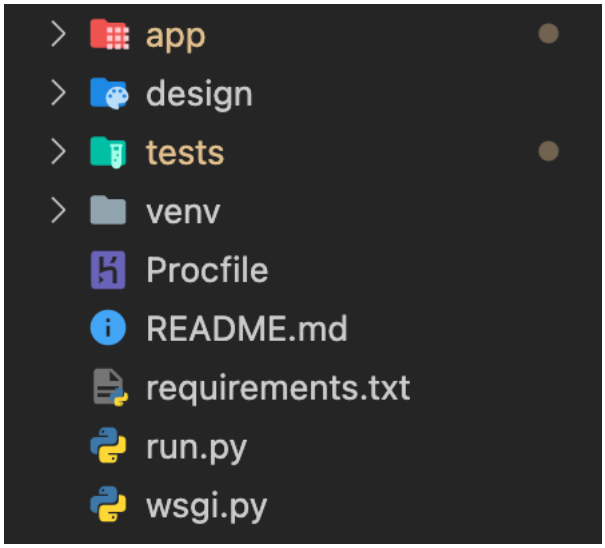

Figure 13: Project Structure

Figure 7 (above) shows the folder structure for this project. All source code is stored in the *'app'* folder, which is currently a python package with its own *'__init__.py'*. All design documents are stored in *'design'* and all tests and testing configurations are stored in *'tests'*. *'Procfile'* is used for my production deployment on the *Heroku* service, it contains a *gunicorn* command that points to my application run script *'wsgi.py'*. Within the app folder, I have all templates (HTML), stylesheets (CSS), backend (Python), API (Python), and database (SQL) files.

Flask App

Table 2: Flask App Initialisation (__init__.py)

```
from flask import Flask, jsonify
from os.path import join, dirname, realpath
from flask_login import LoginManager

# SQLAlchemy for database creation and updating
from flask_migrate import Migrate

'''App initialisation'''
# initialising flask app and path to database
def create_app():
    app = Flask(__name__)

    app.config.from_object('app.config.DevelopmentConfig')

    UPLOAD_FOLDER = join(dirname(realpath(__file__)), 'static/uploads')
```

```
app.config['UPLOAD_FOLDER'] = UPLOAD_FOLDER
# return app

from app.models import db, User
db.init_app(app)
# to handle adding or removing of columns of already created DB
migrate = Migrate(app, db)

@app.route('/ping', methods=['GET'])
def ping_pong():
    return jsonify({
        'status': 'Epic success',
        'message': 'pong!'
        })

from app.admin_views import admin
from app.login_views import login
from app.views import base

'''Log-in manager initialisation'''
# initalising login manager
login_manager = LoginManager()
login_manager.init_app(app)
# link to log in page
login_manager.login_view = "login.logIn"


''' Misc and Error Handling'''
# user loader to remember previously visited users
@login_manager.user_loader
def load_user(user_id):
    return User.query.get(int(user_id))

app.register_blueprint(admin)
app.register_blueprint(login)
app.register_blueprint(base)
return app
```

Table 2 contains all code used to initialise the Flask app. I followed the design pattern of using an *application factory* to create instances of my Flask app which can then be used independently for development, testing and production. Using "app.config.from_object" I can load the chosen application configuration, by default this is the development config. I set the default upload folder to "*static/uploads"* which is where any user uploaded files shall be stored. In the final lines of the above snippet, I initialise the log-in manager and set the default log-in view (to redirect to if a user tries to access a log-in required page). I have also utilised a user loader which will allow users to visit my site multiple times and it will remember their login. I have used *Flask* blueprints to split my web-app into three distinct modules (admin, login, and base). This allows each *routes* python file to be a manageable size.

Database Creation

Table 3: Database Creation Code (models.py)

```
'''DB handling and table creation'''
# initialising database
db = SQLAlchemy()


# user table, to track current registered users that have been approved
class User(db.Model, UserMixin):
    id = db.Column(db.Integer, primary_key = True)
    email = db.Column(db.String(100), unique = True, nullable = False)
    password = db.Column(db.String(80), nullable = False)
    fName = db.Column(db.String(20), nullable = False)
    sName = db.Column(db.String(30), nullable = False)
    userType = db.Column(db.String(20), nullable = False)
    questions = db.relationship('Question', backref='author')
    answers = db.relationship('Answer', backref='author')
    comments = db.relationship('Comment', backref='author')


# question table to track questions being asked on posts
# with foreign keys linking to users and boards
class Question(db.Model):
    id = db.Column(db.Integer, primary_key = True)
    qTitle = db.Column(db.String(500), nullable = False)
    qBody = db.Column(db.String(1000))
    postDate = db.Column(db.DateTime)
    posterId = db.Column(db.Integer, db.ForeignKey(User.id))
    boardId = db.Column(db.Integer, db.ForeignKey(QABoard.id))
```

```
    answers = db.relationship('Answer', backref='question')
    comments = db.relationship('Comment', backref='question')
```

In my web-app I have used *SQLAlchemy* to define and access my database, this allows me to create, read, update, and delete database tables and records without the need for any SQL code. Each class definition directly links to a table within the database. Within the *User* table I have created 'backrefs' which allow me to query foreign key tables without the need for joins. For example, I can retrieve the first name of an author of a question by querying "Question.author.fName" instead of using a join between the Question and User table. I have done the same for the *Board, Answer and Comment* tables.

Database Access (API)

I have developed a simple API to interact between the flask application and the database server/file. One API for User actions and one for Board actions. Creating an API allows me to call these 'helper' functions in multiple places in the web-app which increases code-reuse and optimises the application. By having robustly defined interfaces between the *Flask* app and API, I can easily replace and make changes to either section of code without affecting the main function of the app. It also provides an agnostic design for interacting with different types of databases (In my case, SQLite, and MySQL – Jaws DB). For all functions within the API, I have defined docstrings to allow for better readability and user understanding. Each docstring defines the function's purpose, the arguments, and what it returns.

User Access

Table 4: User Access Code (dbUserAccess.py)

```
'''Class to create, read, update, verify, and delete user profiles'''
class userAccess(object):
    # Function to add user to json DB, requires valid email, password, first name and surname
    def addPendingUser(db, PendingUser, email, password, fName, sName, userType):
        """Function to add user to pendinguser table


        Args:
            db (object): Database object from app.py
            PendingUser (object): PendingUser table object from app.py
            email (string): Unique email of user
            password (string): plain text password
            fName (string): First Name
            sName (string): Surname
            userType (string): User level requested on sign-up
        """
        # Hash the input password
        hash = userAccess.get_hashed_password(password)
        # Convert hash to string
        hash = str(hash)
```

```
    # Extract raw hash
    hash = hash.split("")
    hash = hash[1]
    # Form dictionary to be appended to DB
    user = PendingUser(
        email = email,
        password = hash,
        fName = fName,
        sName = sName,
        userType = userType)
    db.session.add(user)
    db.session.commit()
# Returns nothing, function approves user, i.e. deletes user from PendingUser
def denyUser(db, PendingUser, email):
    """Function denies user access, removes user from pending user table

    Args:
        db (object): Database object from app.py
        PendingUser (object): PendingUser table object from app.py
        email (string): Email of user you want to deny access
    """
    users = userAccess.getPendingUserDetails(PendingUser)
    for user in users:
        if user.email == email:
            db.session.delete(user)
            db.session.commit()
```

The extract above shows a snippet of two functions that are used often in the main app. *addPendingUser* creates a *PendingUser* record and adds it to the database. This is used when a tutor account is created, it is added to the pending user table. The function *denyUser* removed a given user from the *PendingUser* table when an admin denies them from the approval page.

Board Access

Table 5: Board Access Code

```
'''Class to create, read, update, verify, and delete user profiles'''
class boardAccess(object):

    # Function to create board
    def createBoard(db, QABoard, boardName, boardDesc, imgPath):
        """Function to create Q&A Board
```

```
    Args:
        db (object): Database object from app.py
        QABoard (object): Q&A table object from app.py
        boardName (string): A name for the board, to be displayed on board
        boardDesc (string): A description of what will be discussed in this board
        imgPath (string): Path to the image that shall be used for the board's thumbnail
    """
    # Form board object with passed params
    new_board = QABoard(
        boardName = boardName,
        boardDesc = boardDesc,
        backImg = imgPath
    )
    # Add new board to session
    db.session.add(new_board)
    # Commit DB
    db.session.commit()

# Function to delete board
def deleteBoard(db, QABoard, boardId):
    """Function to delete board from Q&A board page

    Args:
        db (object): Database object from app.py
        QABoard (object): Q&A table object from app.py
        boardId (integer): Unique identifier for board that is going to be deleted
    """
    # Get board with passed ID
    board = QABoard.query.filter_by(id=boardId).first()
    # Delete board
    db.session.delete(board)
    # Commit DB
    db.session.commit
```

The board access API follows a similar structure to that of the user access API (create, edit, and delete functions). I import my board models from *models.py* and import *db* from *models.py* which is then used to add and remove records from the database.

HTML

I have chosen to use the Bootstrap framework as a base for my web application's front-end along with varies custom cascading stylesheets (CSS).

Base Template

Table 6: Base HTML Template

```
        <nav class="navbar navbar-expand-md navbar-dark fixed-top bg-dark">
            <a class="navbar-brand" href="#">
                <image src="{{url_for('static',filename = 'images/wmglogo.png')}}" width = "60" height = "50"
class="d-inline-block align-middle" href="{{ url_for('goHome') }}" alt="">
                WMG Teaching Support System
            </a>
            <div class="navbar" id="navbarSupportedContent">
                <ul class="navbar-nav mr-auto">
                    <li class="nav-item">
                        {% block homeLink %}
                        {% endblock %}
                    </li>
                    <li class="nav-item dropdown">
                        {% if current_user.is_authenticated %}
                        <a class="nav-link dropdown-toggle" href="#" id="navbarDropdown" role="button" data-
toggle="dropdown" aria-haspopup="true" aria-expanded="false">
                        {{current_user.fName}}'s Profile
                        </a>
                        <div class="dropdown-menu" aria-labelledby="navbarDropdown">
                            <a class="dropdown-item" href="#">View Profile</a>
                            <a class="dropdown-item" href="#">Settings</a>
                            <div class="dropdown-divider"></div>
                            <a class="dropdown-item" href="{{url_for('logOut')}}">Log Out</a>
                        </div>
                        {%else%}
                        <a class="nav-link dropdown-toggle" href="#" id="navbarDropdown" role="button" data-
toggle="dropdown" aria-haspopup="true" aria-expanded="false">
                        Profile
                        </a>
                        <div class="dropdown-menu" aria-labelledby="navbarDropdown">
                            <a class="dropdown-item" href="#">Settings</a>
                            <div class="dropdown-divider"></div>
                            <a class="dropdown-item" href="{{url_for('register')}}">Create Account</a>
                            <a class="dropdown-item" href="{{url_for('logIn')}}">Log In</a>
```

```
                    </div>
                    {%endif%}
                </li>
            </ul>
        </div>
    </nav>
```

Table 6 shows a snippet of *base.html.* This base template is then inherited in all remaining HTML templates, this allows me to maximise code re-use and enforce consistency between pages. Here I used two variations of the bootstrap navbar to form the basis for all my webpages. I have used jinja selection statements to vary the webpage depending on if a user is logged in or not, which can subsequently customise the webpage for each user. By using jinja statements, I can reference route methods within the *Flask* app which when hosted on a web server, these statements are replaced by HTML links.

Table 7: Search Results HTML

```
<div class="card overflow-auto" style="max-height: 57rem;">
    <div class="card-header text-white bg-dark mb-3">
        Search Results
    </div>
    {% if boards is defined and boards|length > 0 %}
    {% for board in boards %}
    <div class="row">
        <div class="card-body" style="margin-left: 1rem; margin-right: 1rem">
            <div class="card border-dark">
                <div class="card-body">
                    <a href="/Q-A-Board/id/{{ board.id }}"><h5 class="card-title">{{ board.boardName }}</h5></a>
                    <p class="card-text">{{ board.boardDesc }}</p>
                </div>
            </div>
        </div>
    </div>
    {% endfor %}
    {% endif %}
```

The above snippet shows the HTML for my search results page. I use a bootstrap card, like the Q&A board to display the search results. I have passed an array of questions and answers to the HTML file which can then be iterated through using jinja commands. For each question/answer, I display the item's name and description.

CSS

I have used a mixture of inline and external CSS for this project, where padding and text alignment has been executed inline and any other parameters are customised in external CSS files.

Table 8: Home CSS

```
.bg {
  /* The image used */
  background-image: url("images/DAC.jpg");

  /* Full height */
  height: 100%;

  /* Center and scale the image nicely */
  background-position: center;
  background-repeat: no-repeat;
  background-size: cover;
}
```

Table 8 shows a snippet of CSS which is used to display a background image in my web app. It points to an image within the static folder and places it in the centre of the screen. I have utilised similar commands elsewhere to form a custom error page for 404 error messages.

Test Design

Using PyTest I can execute a set a of test cases which link back to my functional requirements and user stories (and their acceptance criteria).

Table 9: conftest.py

```
import os
import sys
import inspect
import pytest

currentdir = os.path.dirname(os.path.abspath(inspect.getfile(inspect.currentframe())))
parentdir = os.path.dirname(currentdir)
sys.path.insert(0, parentdir)

from app import create_app

@pytest.fixture
def app():
```

```
    app = create_app()
    app.config.from_object('app.config.TestingConfig')
    return app
```

Table 9 shows use of the application factory, I create a *pytest fixture* which can be passed to any tests that need it. The fixture creates an app and uses the testing config upon which it then returns the app.

Table 10: Example test case

```
def test_ping(app):
    client = app.test_client()
    resp = client.get('/ping')
    data = json.loads(resp.data.decode())
    assert resp.status_code == 200
    assert 'pong' in data['message']
    assert 'success' in data['status']
```

Table 10 shows a simple test case using PyTest. In the application factory in table 8, I define a default route called *'ping'* which provides a basic endpoint to test. The test case above, creates a test client, performs a GET of the endpoint '/ping' and reads the data from the response. The 'assert' statement evaluates the code following them and return *True* if the code itself is also *True*. If all assert statements pass as true, then the test case passes.

Table 11: Testing Q&A Board Page

```
def login(client, email, password):
    return client.post('/log-in', data=dict(
        userEmail=email,
        userPassword=password
    ), follow_redirects=True)

def logout(client):
    return client.get('/logged-out', follow_redirects=True)

def test_QABoardHome(app):
    client = app.test_client()
    resp = client.get('/Q-A-Board')
    assert b"Q&A Boards" not in resp.data
    assert resp.status_code == 302

    email = 'tutor@warwick.ac.uk'
    password = 'tutor'
    login(client, email, password)
```

```
    resp = client.get('/Q-A-Board')
    assert b"Q&A Boards" in resp.data
    assert resp.status_code == 200
```

The above snippet tests the endpoint ‘Q-A-Board’ using PyTest. I have defined two helper functions, ‘login’ and ‘logout’ which perform login operations as login is required to access the page. A login is performed by posting a dictionary of the user’s email and password. The function ‘test_QABoardHome’ completes two tests, one where the page is accessed without logging in, and one where the user is logged in. In the first test we check that the page accessed is not the Q&A board and check the response code shows a redirect has happened. In the second test we check that the board was accessed successfully.

Continuous Integration

Using *GitHub Actions,* I can run my test cases on every *push* and *merge* into the *‘main’* branch.

Table 12: GitHub Actions YAML

```
name: Flask Test

on:
  push:
    branches: [ main ]
  pull_request:
    branches: [ main ]

jobs:
  build:

    runs-on: ubuntu-latest
    strategy:
      max-parallel: 4
      matrix:
        python-version: [3.8]

    steps:
    - uses: actions/checkout@v2
    - name: Set up Python ${{ matrix.python-version }}
      uses: actions/setup-python@v1
      with:
```

```
      python-version: ${{ matrix.python-version }}

    - name: Install Dependencies
      run: |
        python -m pip install --upgrade pip
        pip install -r requirements.txt

    - name: Run Tests
      run: |
        python -m pytest
```

Table 11 shows my YAML script for the GitHub continuous integration service. It performs 3 actions:

1. Set up python
    a. I use version 3.8, but you can pass multiple python versions to test against.
2. Install dependencies
    a. Upgrade pip
    b. Install requirements.txt
3. Run tests
    a. Run PyTest

Once the action is verified, my *Heroku* instance automatically deploys the tested version to production.
In the future I would look to set up multiple workflows to test more thoroughly and build docker images automatically. It is also possible to test using pylint, to check all code meets style guidelines and PEP8.

Progression of Work, Current Status & Future Work

Progression

At the start of December, I began with a simple, one page, Flask application using the template supplied by the module Tutor. Over time this evolved by including more advanced HTML elements and more Flask endpoints.

```
from flask import Flask
from flask import url_for
from flask import render_template



app = Flask(__name__)


posts = [
    { "title": "The answer for the Session 8 quiz is available on the Moodle", "name": "Young Park", "date": "20 Oct 2021", "count": 2 },
    { "title": "Welcome to WM393 module", "name": "Young Park", "date": "05 Sep 2021", "count": 8 }
```

```
]

@app.route('/')
def index():
    return render_template('list.html', title='Q&A Board', posts=posts, imgPath = "resources/wmg-the-university-of-warwick-logo-vector.png")

if __name__ == '__main__':
    app.run(debug=True)
```

Initially I used a ‘make-shift’ log in system, where I had a loggedIn variable to track user authentication. This proved quite an inefficient method of tracking this and was very inconsistent. It also meant I had to keep passing variables to templates which made my code quite inelegant.

```
userFirstName = 'Jack'

@app.route('/')
def index():
    global loggedIn
    print(loggedIn)
    return render_template('home.html', title='Q&A Board', posts=posts, userFirstName = userFirstName, loggedIn = loggedIn)
    #return render_template('dashboard.html', title='Dashboard Board Title', posts=posts)

@app.route('/logged-out/')
def logOut():
    global loggedIn
    loggedIn = False
    print(loggedIn)
    return 'You are logged out, you will be redirected in 3 seconds', {"Refresh": "3; url = /"}

@app.route('/home')
def goHome():
    return "Hello World"

@app.route('/log-in')
def logInReq():
    global loggedIn
    loggedIn = True
    print(loggedIn)
```

```
    return 'Log In Page'
    #return render_template('logIn.html')


if __name__ == '__main__':
    app.run(debug=True)
```

After some more research I discovered Flask had a built-in log-in manager. This manager would accept a database record and 'login' that user. It allowed for use of decorators like the one shown below. It allowed for me to remove any reference to users within my application, like user details or whether someone was logged in. By using the current_user Flask method (which is accessible globally), I can use user details anywhere in the app with no need for passing parameters into templates.

```
login_manager = LoginManager()
login_manager.init_app(app)
login_manager.login_view = "logInReq"


@login_required
```

Once I had implemented a login manager, I chose to remove my JSON database and start using a SQLite database in conjunction with SQLAlchemy. I created database 'models' that are python classes that define database tables.

```
# database table defnitions
# board table, to track Q&A boards
class QABoard(db.Model):
    id = db.Column(db.Integer, primary_key = True)
    boardName = db.Column(db.String(50), nullable = False)


# user table, to track current registered users that have been approved
class User(db.Model, UserMixin):
    id = db.Column(db.Integer, primary_key = True)
    email = db.Column(db.String(100), unique = True, nullable = False)
    password = db.Column(db.String(80), nullable = False)
    fName = db.Column(db.String(20), nullable = False)
```

By Christmas day, I had a partially working Q&A board, secure login, basic searching, and Heroku hosting. To host on Heroku, all I needed to add was one line of code; a guincorn (web server gateway interface) command which points to my application object within wsgi.py. In this file, I have created an app using my application factory and imported the production config.

```
web: gunicorn wsgi:app
```

```
from app import create_app
```

```
app = create_app()
app.config.from_object('app.config.ProductionConfig')
app.app_context().push()
```

Following on from Christmas day, I continued to improve and enhance the Q&A board using modals and more advanced HTML. For a more in-depth review of the progression please see my git repository.

Work Status

Currently, this version of WMGTSS has a fully functioning version of itself. The following features have been implemented:

- Secure Log-in/out
- Board creation, update, and delete
- Ask a question
- Answer a question

Outstanding/Unimplemented features are as follows:

- Commenting on a question

Plans include completing all missing requirements while also implemented integration with Moodle and the possibility of utilising the University of Warwick SSO (Single Sign-On) system. Following feedback from users, we can implement change requests or fix any bugs. As a first version of my application, I am happy with the design and functionality of it. In future I would like to add more versatility and have a more personalised experience. The main reason for not doing this in the first development is that I did not have the time to I made the decision not to include more due to that. I would also consider including more bespoke design rather than mainly basic Bootstrap design.

Maintenance

By incorporating automated testing, git branches and pull requests, automatic deployment to production and JIRA/Github Projects the project is very easy to manage and maintain. All code is commented clearly, and the goal is to reach 100% comment and testing coverage.

The application's structure is quite modular because it uses many functions and blueprints. This means that further development on this program is relatively easy since code in functions and modules can be edited easily without having to adjust the whole program and interfering with working code. This means that the program is quite maintainable in the future, since fixing errors or changes to each class can be done independently of the main loop. Certain functions can be changed, and this would not affect any important parts of the program.

Conclusion

In conclusion, I have designed, developed, and deployed a semi-production version of WMGTSS using Flask and HTML as my base technologies. I consistently followed standard practice for version control, deployment and web app design while also following the agile manifesto and operating in an agile manner.