Written work Coversheet

Please complete all sections below and attach the completed coversheet to the front of your electronic assignment before submission:

Student Number: (as shown on student ID card) 199110814
Programme: Data Science Degree Apprenticeship
Module Tutor: Stuart Waters
Module Code: DSC5001M
Module Title: Databases
Assignment Title: Creative Artefact
Word Count: 2824

Declaration of Academic Integrity

Please complete before submitting your assignment:

x [x]

By entering an 'x' in the box above, I confirm that I have read and understood the University regulations on cheating and plagiarism and that the work submitted is my own within the meaning of the regulations.

YAHUAS accommodation database

Contents

1. Purpose 3
2. Requirements Specification 3
3. Prototype scope 3
4. Database design 3
4.1 Conceptual design 3
4.2 Logical and Physical design 4
5. Implementation 9
5.1 Dependencies 9
5.2 Web interface design 9
5.3 Webpage design 10
5.4 Data Security 17
6. Code 18

1. Purpose

As set out in the Executive Summary and case study documents, the client - Yorkshire & Humberside University Accommodation Scheme (YAHUAS) - is seeking to centralise accommodation and facilities across the region to optimise on utilisation and lower operating costs.

YAHUAS requires a bespoke centralised database solution available at each university's accommodation office.

A prototype solution is to be developed, focusing on backend database administrative tasks and management reporting via a web interface.

2. Requirements Specification

Components:

A MySQL-compatible database with a backend web interface

- **Database:**
  A full catalogue of student accommodations, lease agreements, and accommodation maintenance details. The full description of data fields is set out in the case study.

- **Web interface:**
  A standard and consistent look for the home page and other follow-up pages. All SQL reports should be presented in tabular form. Correct data validations should be considered for any database manipulations via the interface. Input screens and forms should be created to enable new accommodation information to be recorded.

- **User groups and login functionality:**
  The specified user groups are students, administrators and managers. For the purpose of this prototype, there will be only one user with a dual role of administrator and manager. The user must have login authentication to perform the functions. Administrators should be able to enter data into the system via the web interface.

3. Prototype scope

As the project is in its prototype phase and not near a fully finished product, only partial aspects of the final package will be completed to provide a demonstration of the functionality to expect across the wider final product. The prototype will demonstrate:

- A fully implemented database.
- A navigable back-end website including:
  - User authentication and a login page
  - An example of a report page, featuring joint information drawn from multiple tables
  - An example page for editing database entries, with update, insertion and deletion operations.

4. Database design

4.1 Conceptual design

A relational database model will be used for this product as this is the most common type of database used in industry and as such is an appropriate model to work on for colleagues. Relational

databases have the benefit that data can be maintained in a way that reduces redundancy, administration resource and the risk of error. Data stored in different tables can be joined with one another to link their information together. Data will be organised into *entities*: distinct objects and concepts that each have their own attributes.

4.2 Logical and Physical design

Figure 1 shows the Enhanced Entity Relationship Diagram, listing each entity, its primary keys, its foreign keys and its supertypes and subtypes.

Normalisation

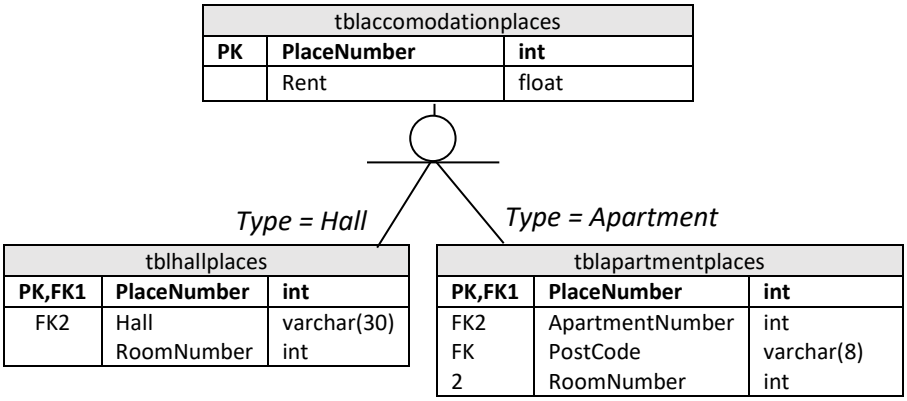
Listed below are the first four normal forms and their requirements.

Normal form	Requirement
First normal form	Each column has a unique name
	Values are atomic
	Columns hold the same datatype
Second normal form	No partial dependencies
Third normal form	No transitive dependencies
Boyce-Codd normal form	No dependencies on non-prime attributes. Dependencies use super keys.

Table 1 – The normal forms from 1NF to BCNF

Each entity table in the entity relationship diagram (Figure 1) has been normalised up to Boyce-Codd normal form. This reduces the amount of redundant data, which in turn reduces file size and helps avoid non-cascading changes whenever a database entry is changed or deleted.

Supertype



tblaccomodationplaces is a supertype of ***tblhallplaces*** and ***tblapartmentsplaces***, and as such the subtype discriminator *Type* will need to be used in cases where *PlaceNumber* as a foreign key – as in ***tblleaseagreements.***

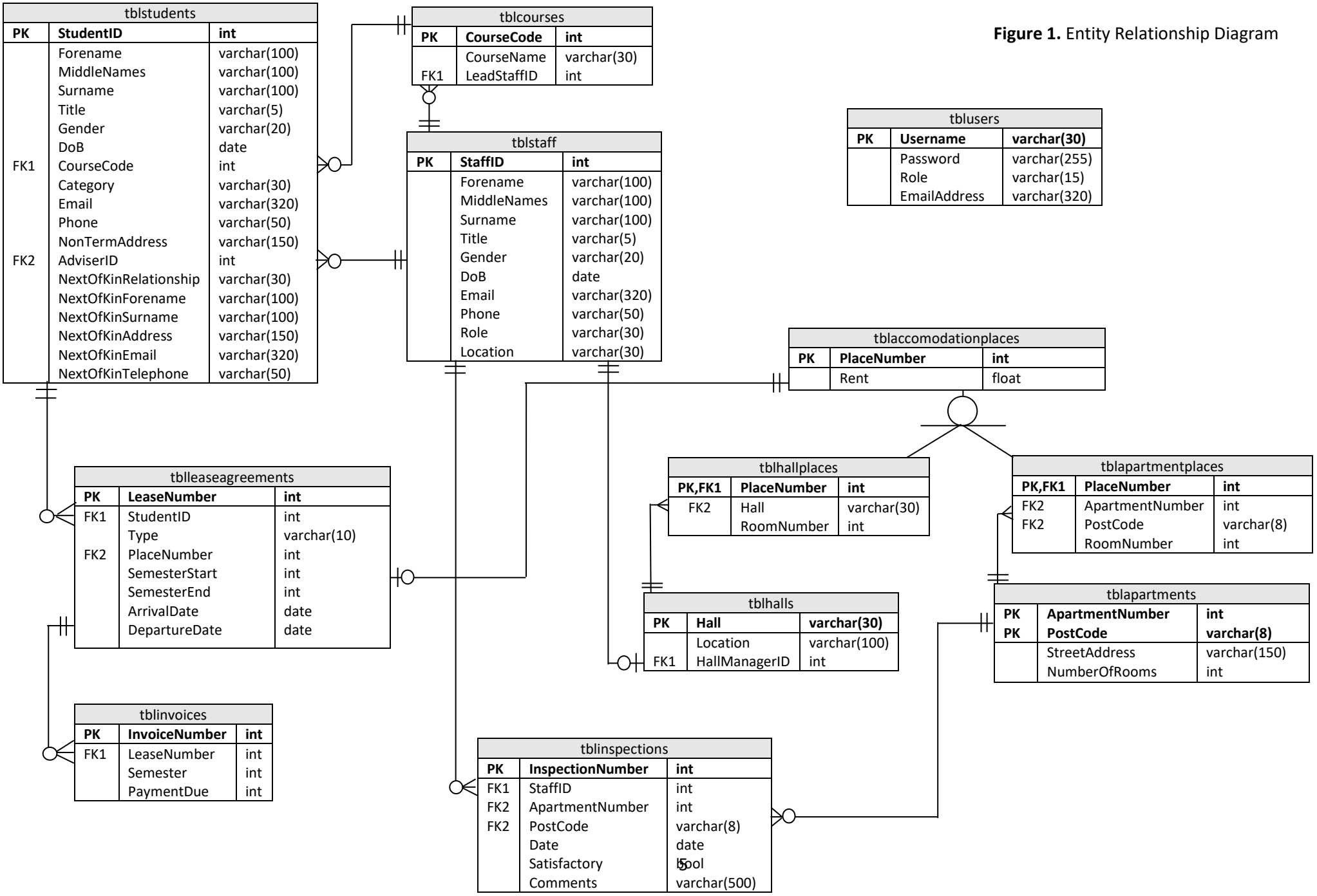


Figure 1. Entity Relationship Diagram

Data attributes

Below are data attribute tables, listing each entity's attributes, their data types, and any additional validation restrictions to be implemented. Please note that not all validation restrictions have been implemented in the prototype version.

tblstudents		
Attribute	**Data type**	**Additional validation restrictions**
StudentID	string	
Forename	string	- 100 character limit
MiddleNames	string	- 100 character limit
Surname	string	- 100 character limit
Title	string	- 5 character limit
Gender	string	- 20 character limit
DoB	date	- After 1900
CourseCode	int	
Category	string	- 30 character limit
Email	string	- 320 character limit; RegEx restriction: email format only;
Phone	string	- 50 character limit; RegEx restriction: phone format only
NonTermAddress	string	- 150 character limit
AdviserID	int	
NextOfKinRelationship	string	- 30 character limit
NextOfKinForename	string	- 100 character limit
NextOfKinSurname	string	- 100 character limit
NextOfKinAddress	string	- 150 character limit
NextOfKinEmail	string	- 320 character limit; RegEx restriction: email format only
NextOfKinTelephone	string	- 50 character limit; RegEx restriction: phone format only

The ***tblstudents*** entity lists the attributes relating to students. One candidate key for the entity is the name of the student, represented as a composite key of the different name fields. However, as a future student may have the same name as an existing student, this could result in non-unique primary key values. As such, the StudentID has been created as an artificial key to serve as the primary key.

tblstaff		
Attribute	**Data type**	**Additional validation restrictions**
StaffID	int	
Forename	string	- 100 character limit
MiddleNames	string	- 100 character limit
Surname	string	- 100 character limit
Title	string	- 5 character limit; from set {Mr., Mrs., Ms., Miss, N/A}
Gender	string	- 20 character limit; from set {Male, Female, Non-Binary}
DoB	date	- After 1900
Email	string	- 320 character limit; RegEx restriction: email format only;
Phone	string	- 50 character limit; RegEx restriction: phone format only;
Role	string	- 30 character limit
Location	string	- 30 character limit

A decision was taken to split Staff and Students into separate entities. While many of the fields between the two are shared and they could have been incorporated into a supertype, for the sake of

simplicity of the prototype these have been treated entirely as two distinct entities. As no entity's foreign keys simultaneously refers to students and staff, this does not cause any issues.

However, in a future version of the website where both students and staff may need login access, the ***tblusers*** entity may use a single foreign key to link to users' personal details (name, email address, etc.) - and this may require the introduction of supertypes and subtypes or an alternative structuring solution.

Tblcourses		
Attribute	**Data type**	**Additional validation restrictions**
CourseCode	int	
CourseName	string	- 30 character limit
LeadStaffID	int	

The ***tblcourses*** entity houses the data about each course. Course code and course name are both candidate keys, but course code has been selected as the primary key as future years may see refreshed courses with the same name running concurrently with the last cohorts of the previous course.

tblleaseagreements		
Attribute	**Data type**	**Additional validation restrictions**
LeaseNumber	int	
StudentID	int	
**Type	string	- 10 character limit
*PlaceNumber	int	
SemesterStart	int	- From set {1, 2, 3}
SemesterEnd	int	- From set {1, 2, 3}
ArrivalDate	date	- After 2020
DepartureDate	date	- After 2020

* *foreign key*

** *subtype discriminator*

The ***tblleaseagreements*** entity contains information about the lease agreements. As pictured in the Entity Relationship Diagram (ERD), the foreign key *PlaceNumber* links to the entity ***tblaccomodationplaces***. Access to the data held in any of its subtypes (***tblapartmentplaces, tblhallplaces***) will require the subtype discriminator *Type*.

tblapartments		
Attribute	**Data type**	**Additional validation restrictions**
ApartmentNumber	int	
PostCode	string	- 8 character limit
StreetAddress	string	- 150 character limit
NumberOfRooms	int	- from set {3, 4, 5}

The ***tblapartments*** entity uses a composite key as its primary key. As only address number and post code are required to identify a unique address, *Street Address* has been left out of the composite key.

tblaccomodationplaces		
Attribute	**Data type**	**Additional validation restrictions**
PlaceNumber	Int	
Rent	float	

tblapartmentplaces		
Attribute	**Data type**	**Additional validation restrictions**
PlaceNumber	int	
ApartmentNumber	int	
PostCode	string	- 150 character limit
RoomNumber	int	

tblhallplaces		
Attribute	**Data type**	**Additional validation restrictions**
PlaceNumber	int	
Hall	string	- 30 character limit
RoomNumber	int	

The ***tblaccomodationplaces*** is a supertype of the subtypes ***tblapartmentplaces*** and ***tblhallplaces***. To reference the data in the subtypes, a subtype discriminator will need to be specified.

tblhalls		
Attribute	**Data type**	**Additional validation restrictions**
Hall	string	- 30 character limit
Location	string	- 100 character limit
HallManagerID	int	

The ***tblhalls*** entity holds data about each hall and relationally links to the *StaffID* of the ***tblstaff*** table for via *HallManagerID*.

tblinvoices		
Attribute	**Data type**	**Additional validation restrictions**
InvoiceNumber	int	
LeaseNumber	int	
Semester	int	- from set {1, 2, 3}
PaymentDue	int	

The ***tblinvoices*** holds information on invoices and references the relevant *LeaseNumber* in the ***tblleaseagreements*** entity.

tblinspections		
Attribute	**Data type**	**Additional validation restrictions**
InspectionNumber	int	
StaffID	int	
ApartmentNumber	int	
PostCode	string	- 8 character limit
Date	date	- After 2020
Satisfactory	bool	
Comments	string	- 500 character limit

The ***tblinspections*** entity introduces an inspection number to act as a unique ID to act as the primary key. Alternatively, a composite key of date, apartment number and post code could be used.

tblusers		
Attribute	**Data type**	**Additional validation restrictions**
Username	string	- 30 character limit
Password	string	- 255 character limit
Role	string	- 15 character limit
EmailAddress		- 320 character limit

The ***tblusers*** entity is not relationally linked to any of the other entities as it is only used to authenticate administrator users and does not contain data about YAHUAS accommodation. As discussed above, in a future iteration of the product when the userbase becomes wider, there is an argument for replacing the EmailAddress attribute with a foreign key to the existing staff and/or student data.

5. Implementation

5.1 Dependencies

The following software versions were used in the development of the prototype:

- XAMPP 8.1.2
  - MariaDB 10.4.22 *(MariaDB is an open-source fork of MySQL)*
  - PHP 8.1.2
  - Apache 2.4.52
- Mozilla Firefox 98.0.2
- HTML5
- CSS 2.1

5.2 Web interface design

The web interface is delivered to the browser as HTML pages generated from server-side PHP via HTTP responses.

The initial approach to coding the website was a procedural paradigm, with a relatively flat file structure.

To improve the modularity, scalability and ease of updates and bug fixes for the website, this approach has begun to shift towards Model View Controller (MVC) object-oriented paradigm. However, due to time constraints, this has only been implemented for the login functionality. In a future version of the prototype, this would be rolled out across all pages and functions.

This newer approach has moved away from using the MySQLi() interface and towards using PHP's PDO object type for connection the database. PDO() is proposed over the alternative MySQLi() as PDO works with 12 different database systems, whereas MySQLi will only work with MySQL databases. This will future-proof the system against potential future migrations to other database systems. However, MySQLi an older, more commonly used standard, so if the development team require it MySQLi can be used in the project in place of PDO.

Database connection

The database handle class, *dbh*, handles connections to the database. This uses private database login credentials and protected functions. This ensures that only the model classes that extend the *dbh* class have access to its connection methods, and the sensitive credential attributes themselves cannot be accessed outside of *dbh*.

5.3 Webpage design

Templating

Two PHP files, '*header.php*' and '*footer.php*' are imported at the top and bottom of every major page to render the same main HTML navigation and main page structure on every page. Along with the CSS templating, this gives the site a consistent look across each page.

Accessibility

The page has been designed with ARIA standards in mind for accessibility, and semantic tags have been used where possible instead of generic *div* tags etc.

Login page

LOGIN DETAILS:

Username: admin
Password: admin

In order to access any part of the website, users need to log in. Trying to access any part of the website without having logged in will result in PHP conditions in the header template redirecting the user to the log in page. Once the user has successfully logged in, a PHP Session will start and persist between pages. Upon logout, the Session is destroyed and the user will have to log in again.

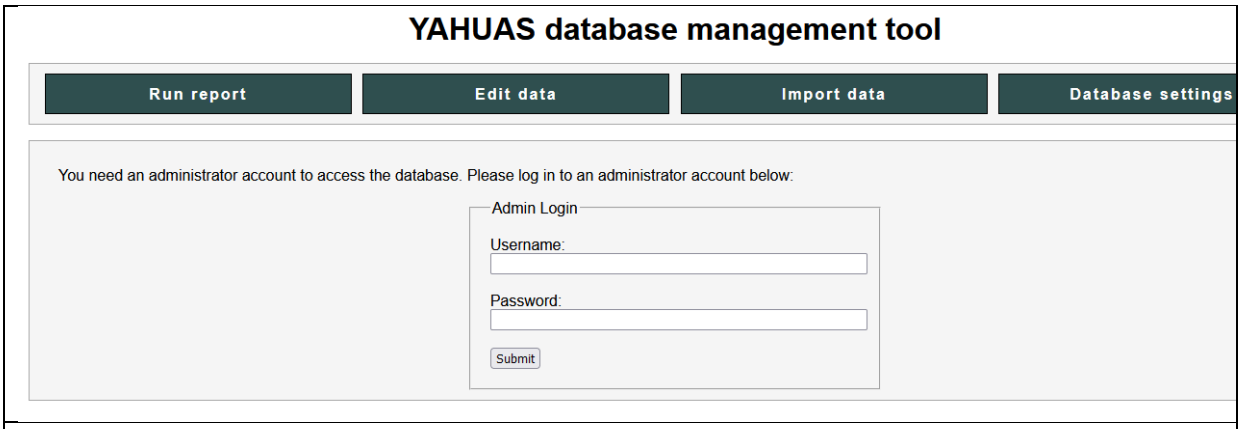


Figure 2. The management tool login page.

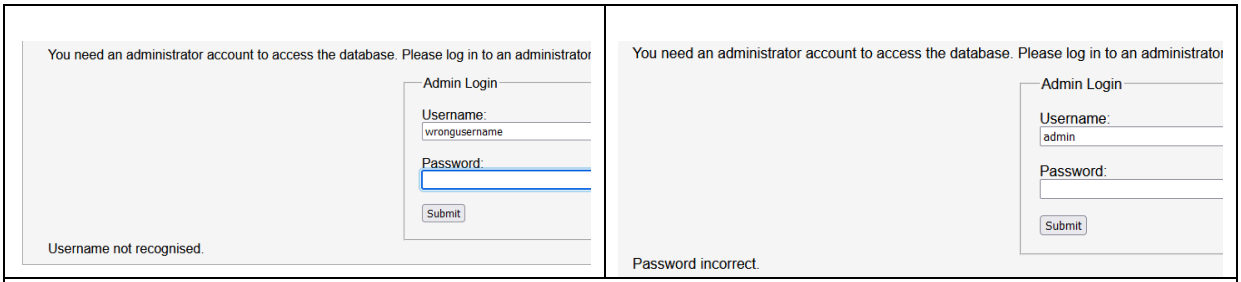


Figure 3. Error messages displayed after the submission of a) an unrecognised username and b) and incorrect password.

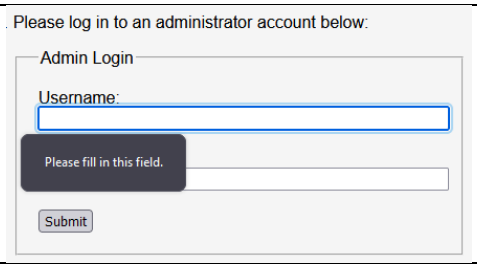
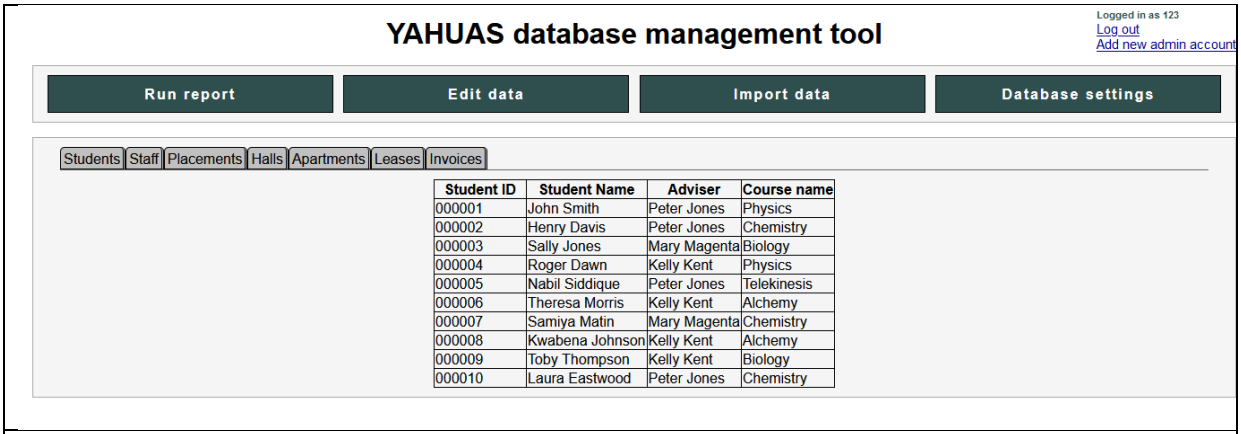


Figure 4. Users are prevented from submitting blank login details by the HTML input *required* parameter.

Report page

Upon successful login, the user is redirected to the index page. This is the report page for generating and displaying tables of data selected from the database.



Student ID	Student Name	Adviser	Course name
000001	John Smith	Peter Jones	Physics
000002	Henry Davis	Peter Jones	Chemistry
000003	Sally Jones	Mary Magenta	Biology
000004	Roger Dawn	Kelly Kent	Physics
000005	Nabil Siddique	Peter Jones	Telekinesis
000006	Theresa Morris	Kelly Kent	Alchemy
000007	Samiya Matin	Mary Magenta	Chemistry
000008	Kwabena Johnson	Kelly Kent	Alchemy
000009	Toby Thompson	Kelly Kent	Biology
000010	Laura Eastwood	Peter Jones	Chemistry

Figure 5. Basic data from the tblstudent, tblstaff and tblcourses entities is joined by inner joins and displayed in tabular format.

```
/* Join the Student table via its foreign keys with Staff and Courses */
$sql = "SELECT tblstudents.*, tblstaff.Forename AS StaffForename, tblstaff.Surname AS StaffSurname, tblcourses.CourseName"
. " FROM tblstudents "
. " INNER JOIN tblstaff "
. " ON tblstudents.AdviserID=tblstaff.StaffID"
. " INNER JOIN tblcourses "
. " ON tblstudents.CourseCode=tblcourses.CourseCode"
. " ORDER BY tblstudents.StudentID ASC;";
```

Figure 6. PHP-generated SQL code to used to select and join the data in the previous figure.

Please note: in the prototype version, only the basic report for Students has been completed so far.

Edit page

The edit page can be accessed through the large navigation bar at the top of the site. This page enables a user to create, edit and delete entries in the database.

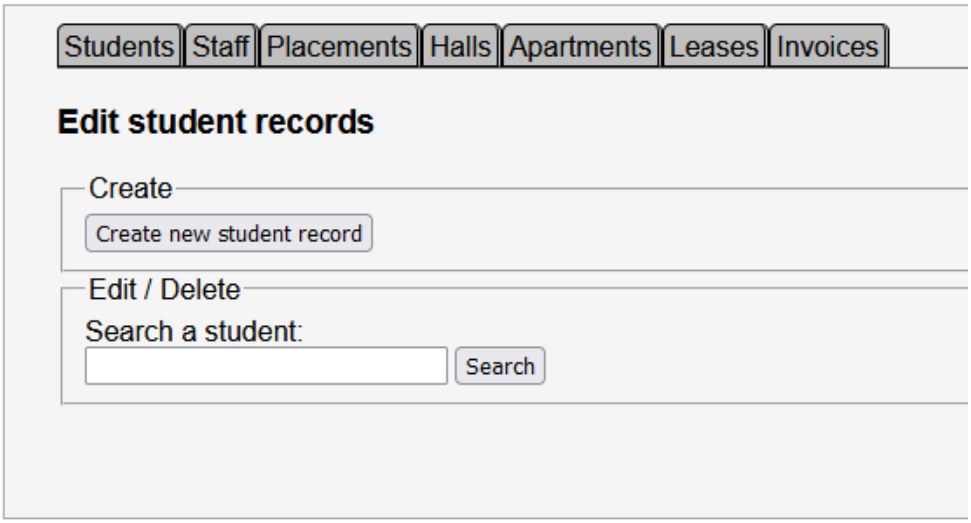


Figure 7. The page for editing records.

When the webpage is being prepared, the server connects to the database and gets a list of all possible values. In the case of students it retrieves the ID, forename and surname of each student. These are concatenated into datalist options to make it easy for the user to find the relevant search result: they can use the student name *or* the student ID and autocomplete suggestions will update as they continue to type.

```
<datalist id="lookup" name="lookup">
    <?php
    /* Datalist options provided are a concatination of the Student ID (6 digit format), Forename and Surname.
    while ($row = mysqli_fetch_array($rs)){
        echo ('<option>'
            . '(' . sprintf("%06d", $row['StudentID']) . ') ' . $row['Forename']. ' ' . $row['Surname'])
            . '</option>';
        }
    ?>
</datalist>
```

Figure 8. Datalist options are taken from the database.

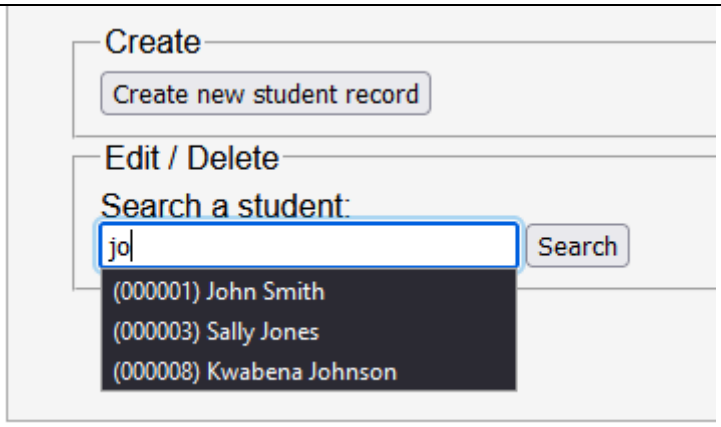


Figure 9. When a user starts typing, the form suggests a list of names that match the characters.

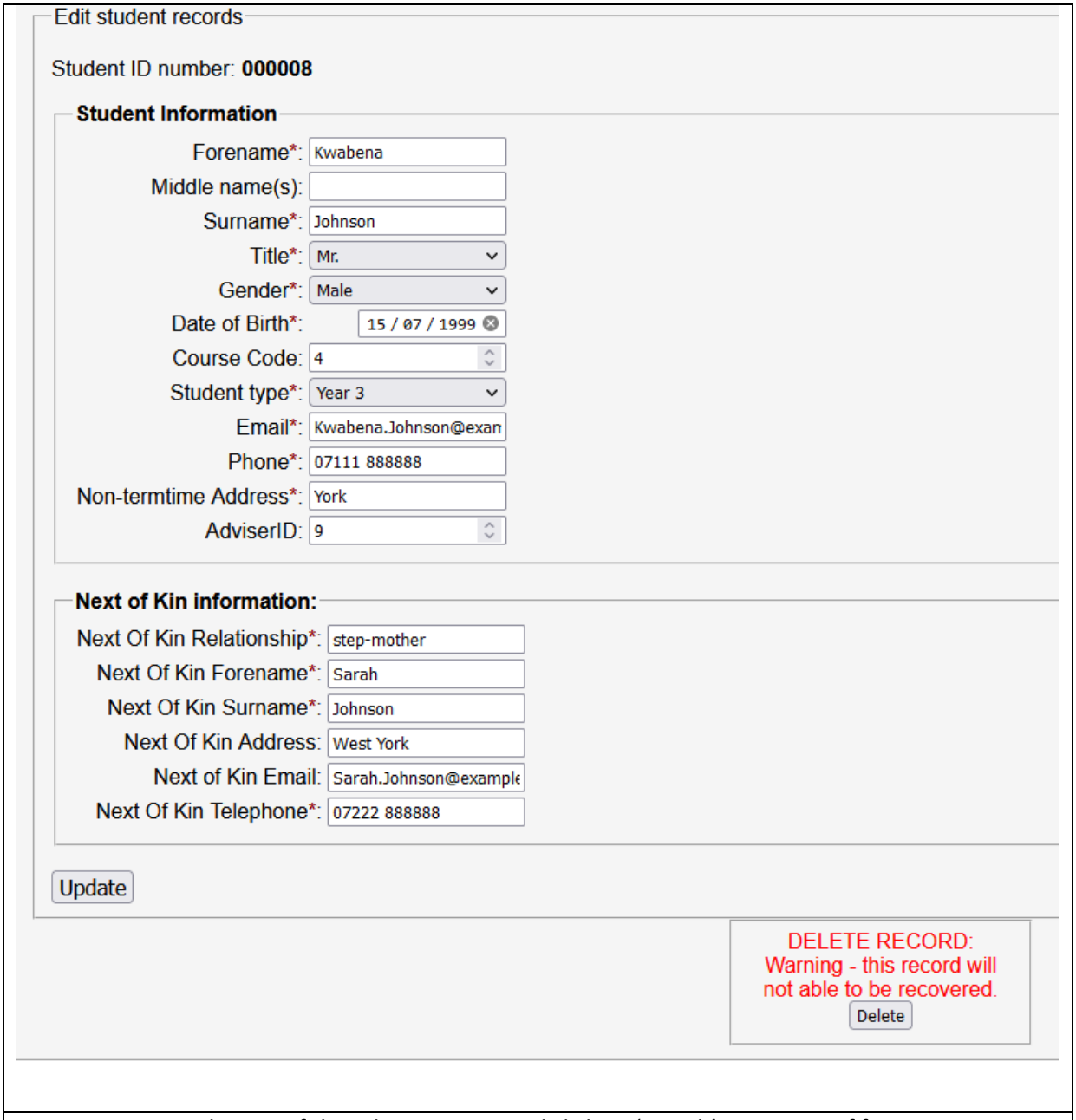


Figure 10. Upon selection of the relevant name and clicking 'Search', a new set of forms are generated that read in values from the database. Upon clicking update they are updated in the database. HTML input *required* parameters prevent the user from submitting empty fields when they are mandatory.

```
/* Update all values from the form in the database. */
$sql = "UPDATE tblstudents SET "
    . "Forename = '" . $_REQUEST['Forename'] . "', "
    . "MiddleNames = '" . $_REQUEST['MiddleNames'] . "', "
    . "Surname = '" . $_REQUEST['Surname'] . "', "
    . "Title = '" . $_REQUEST['Title'] . "', "
    . "Gender = '" . $_REQUEST['Gender'] . "', "
    . "DoB = '" . $_REQUEST['DoB'] . "', "
    . "CourseCode = '" . $_REQUEST['CourseCode'] . "', "
    . "Category = '" . $_REQUEST['Category'] . "', "
    . "Email = '" . $_REQUEST['Email'] . "', "
    . "Phone = '" . $_REQUEST['Phone'] . "', "
    . "NonTermAddress = '" . $_REQUEST['NonTermAddress'] . "', "
    . "AdviserID = '" . $_REQUEST['AdviserID'] . "', "
    . "NextOfKinRelationship = '" . $_REQUEST['NextOfKinRelationship'] . "', "
    . "NextOfKinForename = '" . $_REQUEST['NextOfKinForename'] . "', "
    . "NextOfKinSurname = '" . $_REQUEST['NextOfKinSurname'] . "', "
    . "NextOfKinAddress = '" . $_REQUEST['NextOfKinAddress'] . "', "
    . "NextOfKinEmail = '" . $_REQUEST['NextOfKinEmail'] . "', "
    . "NextOfKinTelephone = '" . $_REQUEST['NextOfKinTelephone'] ."' "
    . "WHERE StudentID = '" . $_REQUEST['StudentID'] ."';" ;
```

Figure 11. The PHP-generated SQL for updating the database with the values submitted in the form.

At the bottom of each search record is a ‘Delete’ button for the deletion of the searched and successfully found student entry.

```
$sql = "DELETE FROM tblstudents WHERE "
    . "StudentID = '" . $_REQUEST['StudentID'] ."';" ;
```

Figure 12. The PHP-generated SQL for deleting a searched entry from the tblstudents table.

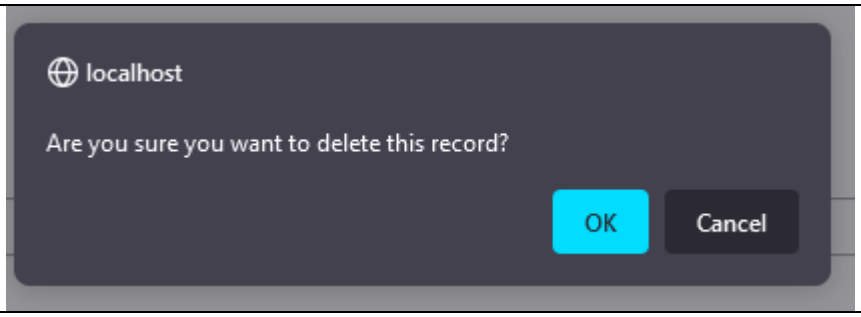


Figure 13. A Javascript confirmation dialogue box pops up when the user clicks ‘Delete’ to make sure the user understands what they are doing and have not clicked by accident.

Users can also click the ‘Create new student record button at the top of the page to create a brand new entry in the database table.

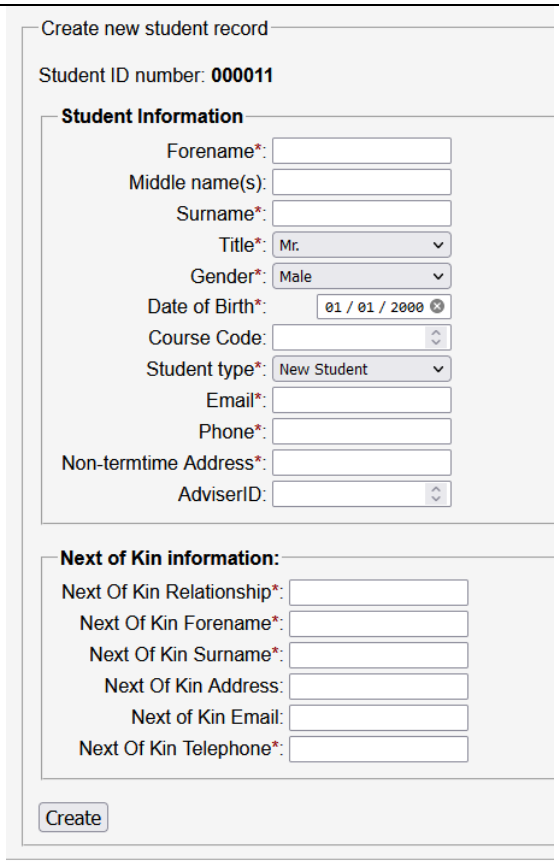


Figure 14. The “Create” view of the editing page. Inputs are left blank for the user to populate.

```
/* Insert given values into the student table as a new entry. */
$sql = "INSERT INTO tblstudents VALUES ( "
    . "'" . $_REQUEST['StudentID'] . "', "
    . "'" . $_REQUEST['Forename'] . "', "
    . "'" . $_REQUEST['MiddleNames'] . "', "
    . "'" . $_REQUEST['Surname'] . "', "
    . "'" . $_REQUEST['Title'] . "', "
    . "'" . $_REQUEST['Gender'] . "', "
    . "'" . $_REQUEST['DoB'] . "', "
    . "'" . $_REQUEST['CourseCode'] . "', "
    . "'" . $_REQUEST['Category'] . "', "
    . "'" . $_REQUEST['Email'] . "', "
    . "'" . $_REQUEST['Phone'] . "', "
    . "'" . $_REQUEST['NonTermAddress'] . "', "
    . "'" . $_REQUEST['AdviserID'] . "', "
    . "'" . $_REQUEST['NextOfKinRelationship'] . "', "
    . "'" . $_REQUEST['NextOfKinForename'] . "', "
    . "'" . $_REQUEST['NextOfKinSurname'] . "', "
    . "'" . $_REQUEST['NextOfKinAddress'] . "', "
    . "'" . $_REQUEST['NextOfKinEmail'] . "', "
    . "'" . $_REQUEST['NextOfKinTelephone'] ."' "
    . ");" ;
```

Figure 15. The PHP-generated SQL code that carries out the database action upon clicking ‘Create’.

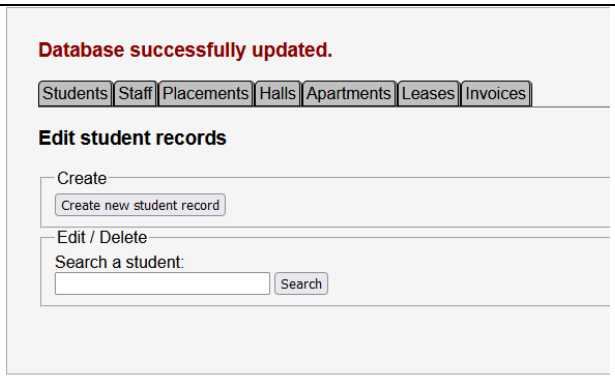


Figure 16. Upon any successful operation (update, insert or delete), an message is displayed confirming success of the operation.

User account creation page

For ease, a user account creation page has been developed. In the final version this will only be available to administrators.

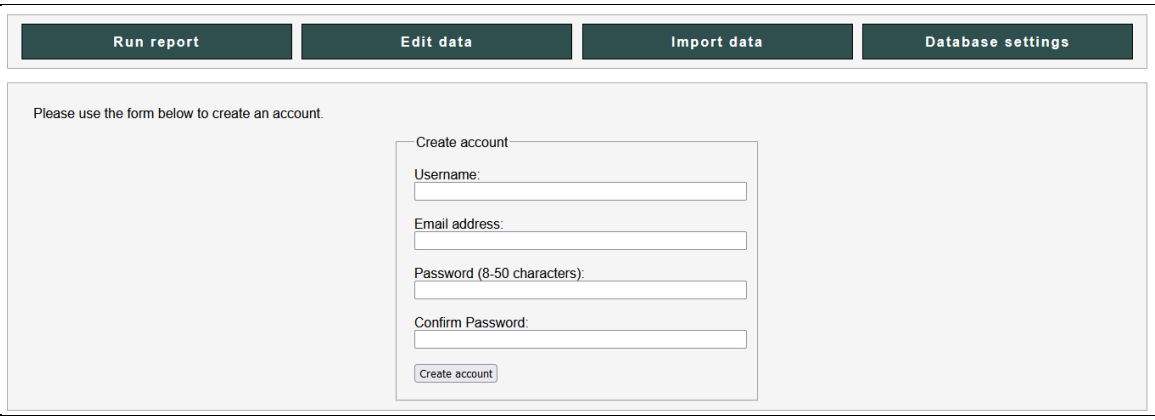


Figure 17. The user account creation page.

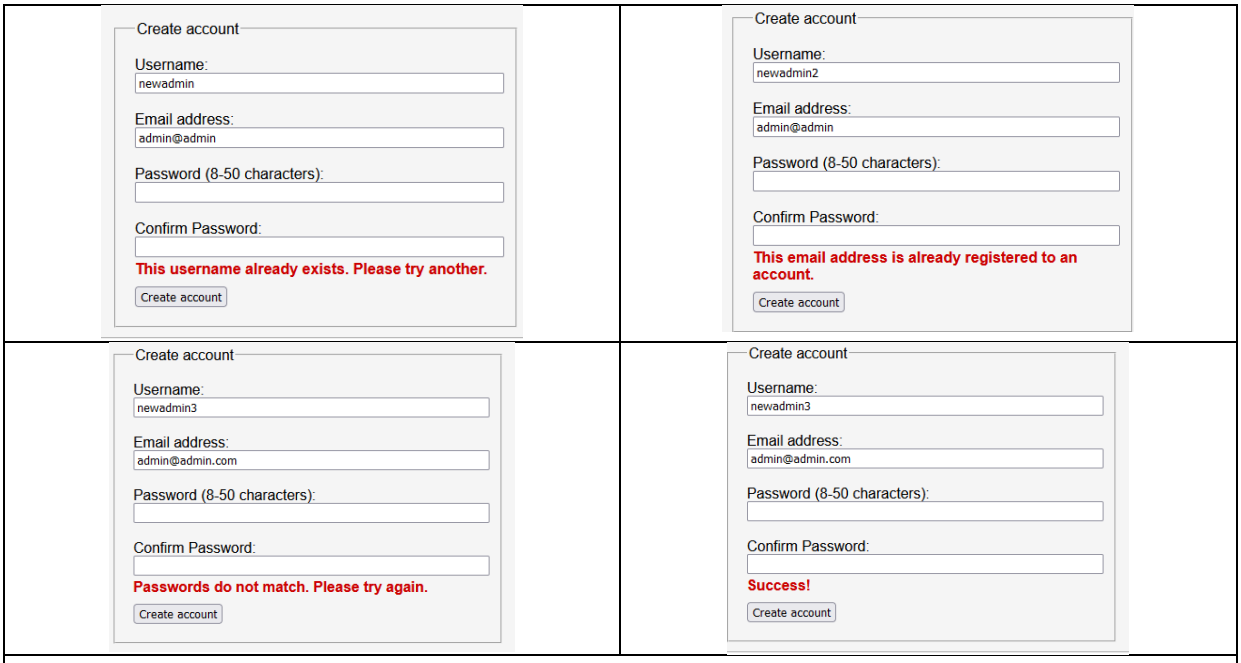


Figure 18. Messages displayed to the user after different conditions are triggered.

```php
/* Check if username already exists in database */
$sql = "SELECT * FROM tblusers WHERE Username='$username' LIMIT 1";
$rs = $conn->query($sql);
$username_present = $rs->num_rows;

/* Check if email address already exists in database */
$sql = "SELECT * FROM tblusers WHERE EmailAddress='$email_address' LIMIT 1";
$rs = $conn->query($sql);
$email_present = $rs->num_rows;
```

Figure 19. Checking if a username or email address already exists in the user account table.

Database Setting Page

Finally, a proposed Database Settings page has been developed. This is to aid in the yearly maturation of Year 1 students to Year 2, Year 2 students to Year 3, etc. and makes use of the MySQL Event Scheduler functionality. Instead of having to write queries to manually alter all the student year data, the MySQL Event Scheduler can schedule automatic rules that will trigger at a certain time and date for a given period. This page is not yet fully functional, but it reads values from *preferences.php* and will allow users to submit new values, which will simultaneously send an SQL query to the database to update the scheduler rules and change the details logged in *preferences.php.* This php file will be changed into a json file in a future iteration of the product.

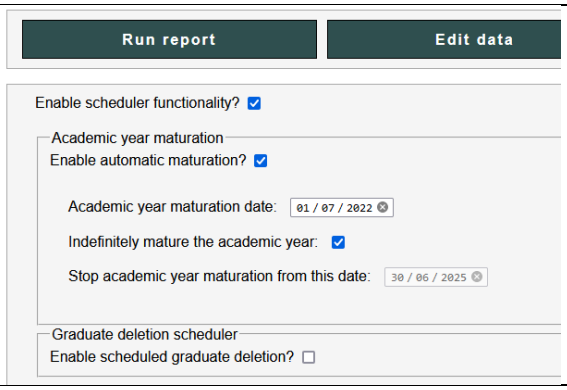


Figure 20. The Database Settings page featuring the scheduler options.

5.4 Data Security

The data stored within the database includes a lot of sensitive person information and care should be taken when storing and handling it. Data retention should fall in line with GDPR laws and data should be stored in a secure place.

All passwords in the database have been hashed using PHP's inbuilt hashing function. Salting is depreciated in the latest PHP version so it has not been necessary to include this. The project code has been written for the current version of PHP which uses the CRYPT_BLOWFISH algorithm as default. Blowfish hashed strings are 60 characters long, but the character length in the database has been extended to 255 as the PHP documentation recommends databases use 255 character string lengths for the password field to future-proof them for future hashing algorithm updates

The next step in improving the security of the YAHUAS tool will be to ensure that the code is protected against SQL injection and cross-site scripting (XSS) attacks to prevent the theft or corruption of sensitive data.

Where possible, validation should be done both on the server side and the client side in the browser.

6. Code

Project code available at: https://github.com/YSiJ-199110814/YAHUAS

Bibliography

McFadyen, R., 2020. *12.4: Mapping Supertypes and Subtypes to a Relational Database*. [online] Engineering LibreTexts. Available at: <https://eng.libretexts.org/Bookshelves/Computer_Science/Databases_and_Data_Structures/Book%3A_Relational_Databases_and_Microsoft_Access_(McFadyen)/12%3A_Appendix_B_-_Supertypes_and_Subtypes/12.04%3A_Mapping_Supertypes_and_Subtypes_to_a_Relational_Database> [Accessed 6 June 2022].

W3schools.com. n.d. *PHP MySQL Connect to database*. [online] Available at: <https://www.w3schools.com/php/php_mysql_connect.asp> [Accessed 6 June 2022].

Php.net. n.d. *PHP: crypt - Manual*. [online] Available at: <https://www.php.net/manual/en/function.crypt.php> [Accessed 6 June 2022].

Php.net. n.d. *PHP: SQL Injection - Manual*. [online] Available at: <https://www.php.net/manual/en/security.database.sql-injection.php> [Accessed 6 June 2022].

KistenS, 2022. *Cross Site Scripting (XSS) Software Attack | OWASP Foundation*. [online] Owasp.org. Available at: <https://owasp.org/www-community/attacks/xss/> [Accessed 6 June 2022].