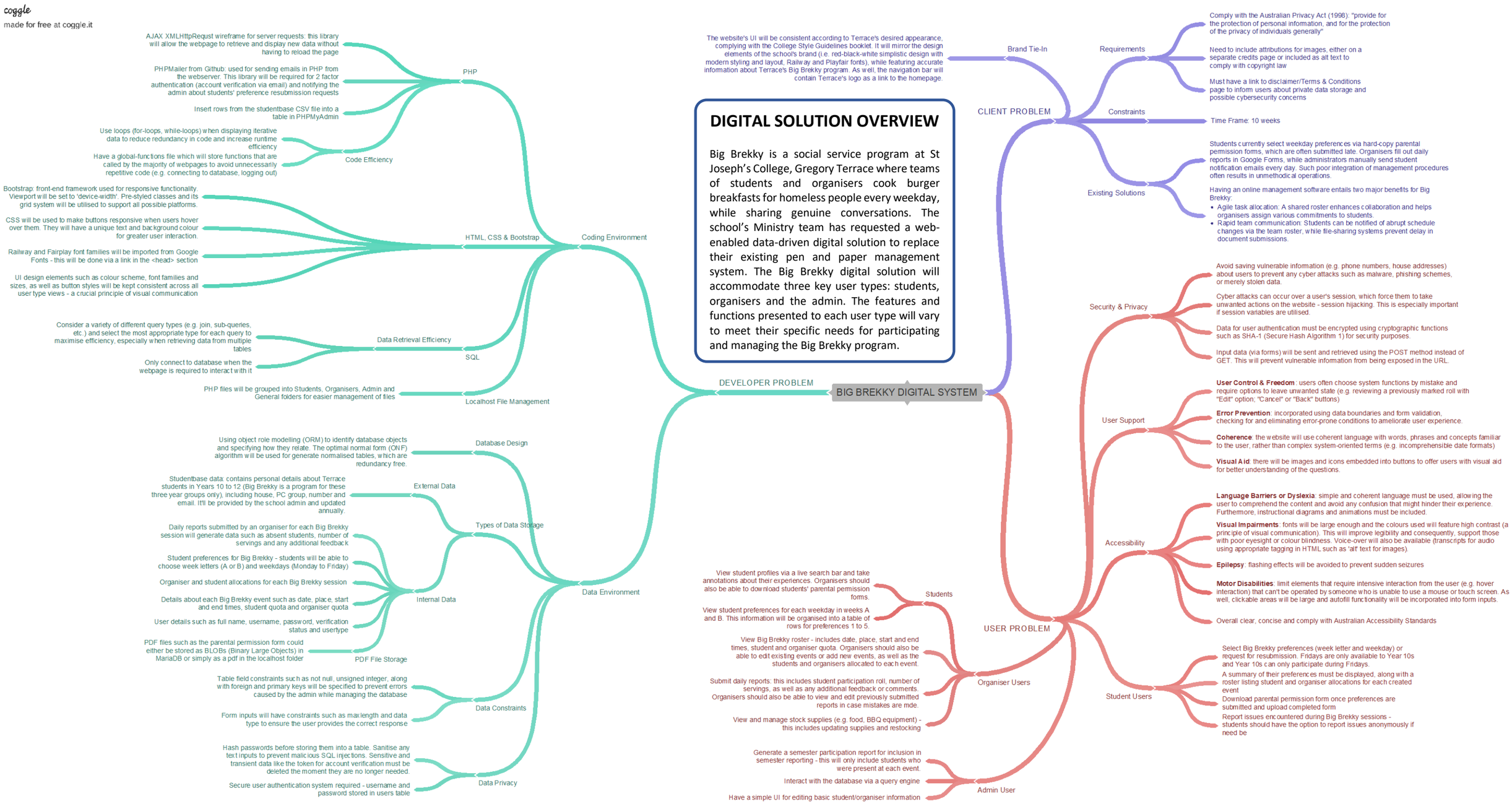
TABLE OF CONTENTS

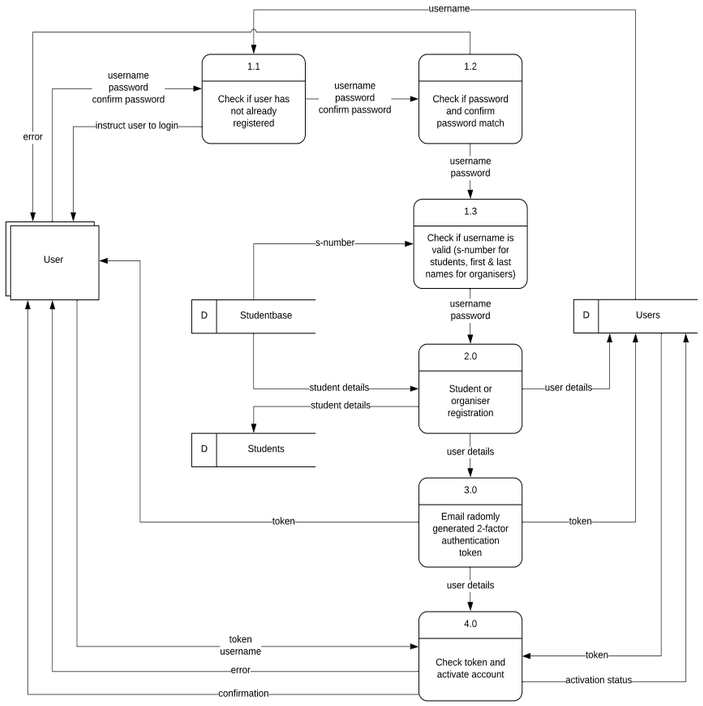
1.0 EXPLORATION OF PROBLEMS



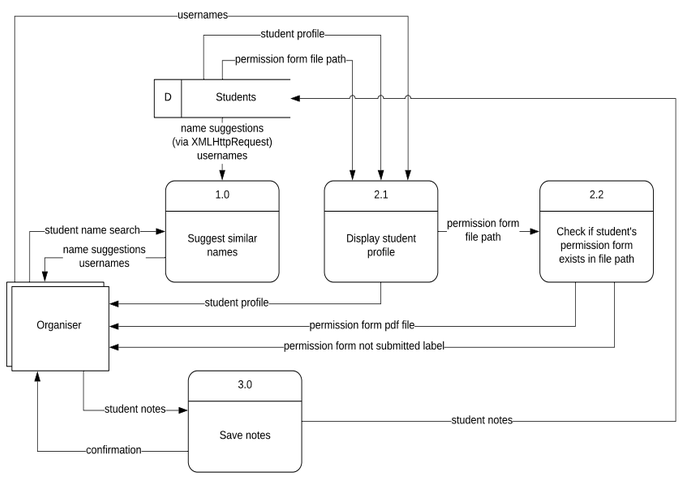
2.0 DESIGN

2.1 DATA FLOW DIAGRAMS

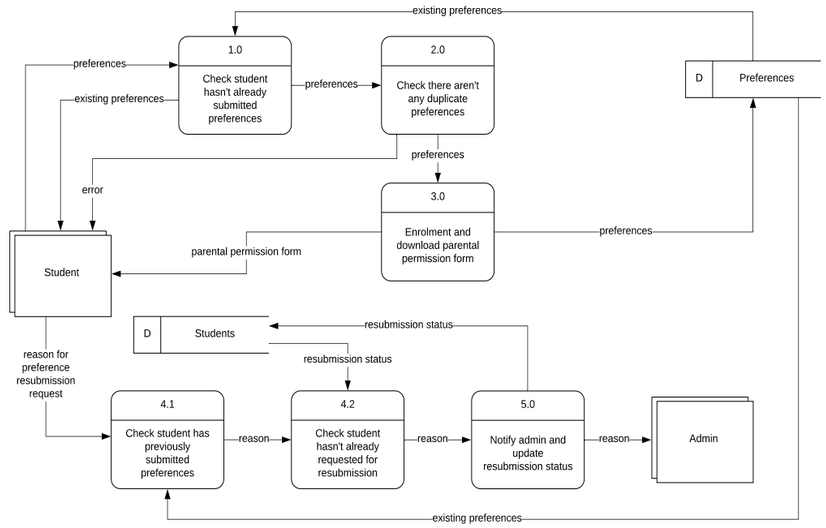
USER REGISTRATION



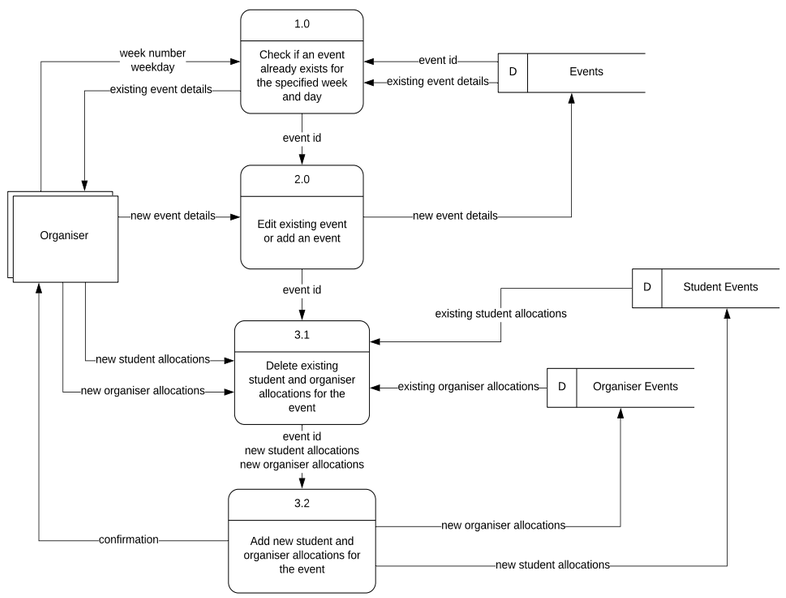
STUDENT PROFILE MANAGEMENT (ORGANISER)



BIG BREKKY ENROLMENT (STUDENT)



EDIT ROSTER (ORGANISER)



2.3 PRESCRIBED & SELF-DETERMINED CRITERIA

PRESCRIBED CRITERIA

- Student actions:
  - [PC1] Select weekday preferences for Big Brekky enrolment and view selected preferences
  - [PC2] Needs the ability to edit preferences
  - [PC3] Download electronic copy of the parental permission form
- Organiser actions:
  - [PC4] View event summaries – include date, type and location, quota and current registration levels of participants
  - [PC5] Apply participation caps or encourage more to get involved
  - [PC6] View and manage students' parental permission form
  - [PC7] Print out an attendance roll and complete it online later when a digital device is accessible
  - [PC8] Annotate student profiles with performance notes
- Admin actions:
  - [PC9] Upload user data and promote users to organisers
  - [PC10] Generate semester participation reports for inclusion in semester reporting
- Data requirements:
  - [PC11] Admin can add cohort studentbase data by uploading an appropriate .csv file. This dataset consists: s-number, first & last names, email address, house & PC group
  - [PC12] Rostered sessions have a minimum of: date, street address, start and end times, attendance cap, minimum required no. of organisers
- User experience:
  - [PC13] Comply with Australian Accessibility Standards and the College Style Guidelines booklet
  - [PC14] Comply with the Australian Privacy Act (1998) – ensure that only authorised personnel can view personal data and inform users of their data being stored
  - [PC15] Include appropriate attribution to data and images used. Must comply with copyright law
  - [PC16] Sanitise user input to prevent SQL injection attacks and securely store student information and passwords. Implement 2-factor authentication for secure registration

SELF-DETERMINED CRITERIA

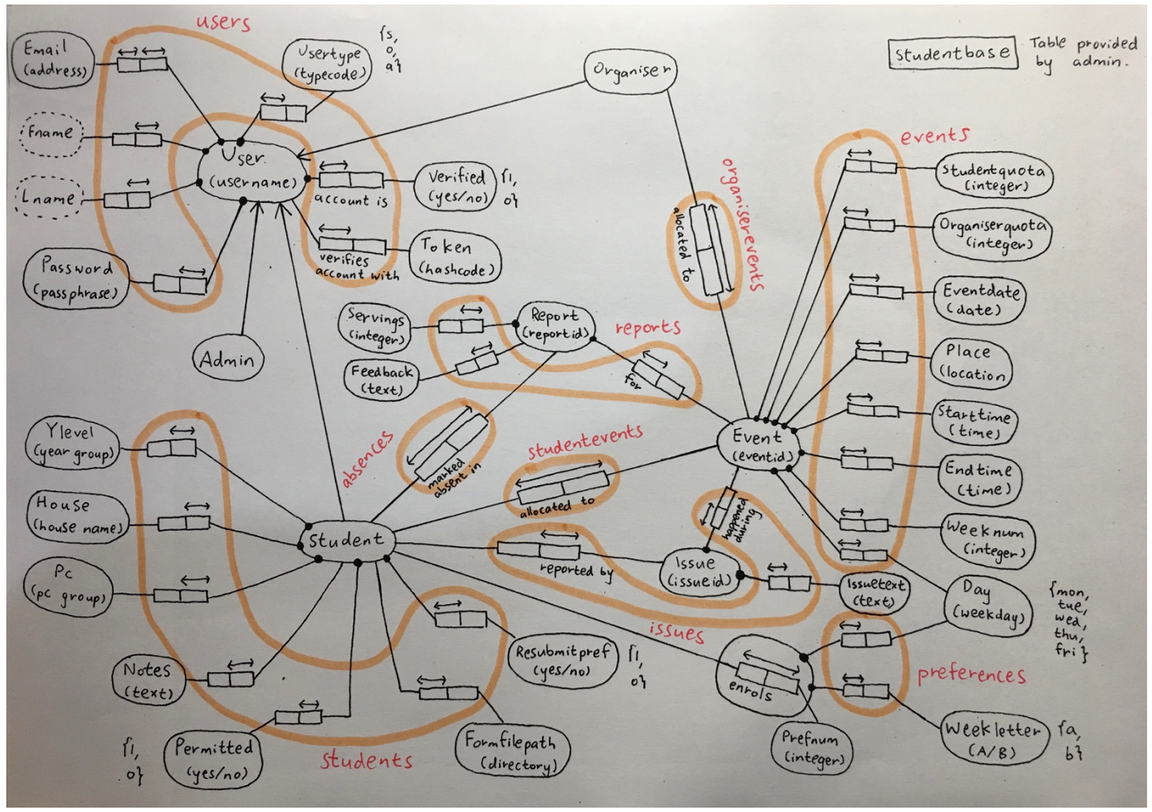
- Student actions:
  - [SC1] Upload a digital copy of their completed parental permission form securely
  - [SC2] Report issues encountered during Big Brekky sessions (anonymously if need be)
- Organiser actions:
  - [SC3] Edit every aspect of the roster, including event details and the students & organisers allocated to each event. Organisers should also have the ability to add new events.
- Admin actions:
  - [SC4] Building a UI for admin to interact with the database instead of having to visit phpMyAdmin
  - [SC5] Edit any aspect of the database via an SQL query engine
  - [SC6] Admin is notified via email when the student submits a request for preference resubmission
- [SC7] Automatically generate a recommended term roster based on student preferences to reduce the workload of the organisers' when managing the roster
- [SC8] Implement AJAX XMLHttpRequests for live database queries and webpage content updates without having to reload the page
- [SC9] UI is compatible with mobile platforms (responsive) and appealing to the human eye
- [SC10] Generation of efficient programming components, data elements and user interface
- [SC11] Error-proofing mechanisms to prevent defects caused by users
- [SC12] Compliant with useability principles (effectiveness, accessibility, safety)
- [SC13] Consider personal, social and economic implications to identify risks of the Big Brekky digital system

Self-determined criteria were constructed during problem analysis (mind map in section 1.0) and updated throughout the development process.

2.4 CONSTRAINTS

- **Time**: digital solution must be completed within the time frame provided (10 weeks). Hence, this prototype will have limitations with regards to demonstrating functionality and responsivity.
- **Accuracy of content**: showcased information about each Big Brekky event and similar references to the Spiritual Retreat Program may not be entirely reflective of Terrace's desired content as this is merely a prototype/
- **Fake data**: studentbase data exported from the School Management System was the only real data used for this digital solution. All other data were generated arbitrarily.
- **Developer experience**: certain functionalities such as "Forgot my password" and secure file uploads may not be implemented due to lack of required programming and web-developing knowledge
- **Legal constraints**: images used will be acknowledge as per the stated licenses to comply with copyright laws

2.2 DATA MODEL USING ORM AND ONF



OBJECT ROLE MODELLING & OPTIMAL NORMAL FORM ALGORITHM

As shown in the ORM diagram, mandatory constraints and relationships have been determined for each role to design a suitable database for the Big Brekky digital system. Student, mentor and admin objects are all subtypes of the user object. All registered users have the attributes 'verified' and 'token'. The token is passed as a GET variable when users click the account verification link via their 2-factor authentication (2FA) emails, allowing the website to identify the user from the URL. Once the account is verified, the 'token' and 'verified fields will be updated to null and 1 respectively.

All 'yes/no' type objects will be stored as booleans (1 or 0) instead of varchars for easily incorporating the values into true/false statements in PHP without requiring any data conversions. The underlying approach to this digital problem was to allow students to only choose week letters and weekdays as their preferences (Approach A), instead of enrolling for a specific event (Approach B). This was because Approach B required the assumption that organisers would have already added all the upcoming events into the database prior to students' preference selection. This is a highly unrealistic assumption and is a demanding task for the organiser in real life. Furthermore, in Approach B, priority is given to students who enrol the quickest, which may potentially result in a large percentage of the slower students being forced to choose commonly disfavoured dates (e.g. a few days before exam block). On the other hand, Approach A is a fairer system for the students, as everyone's preference selections are taken into consideration before generating a term roster (i.e. no student has priority over another).

In the original ORM (Appendix 1), email was used instead of username as the unique identifier for users. However, It was later realised that identifying users with email is a less efficient method requiring more characters being stored in the database table, since all Terrace emails also contain usernames (s-number for students and first & last names for organisers) before the '@terrace.qld.edu.au' domain. Moreover, a lot more fields were added to the events table from the initial database design to provide organisers with full control over events, including student and organiser quotas, as well as variable start and end times of each session. The 'Stock' object from the initial ORM was deemed unnecessary for this prototype as it was an inessential component, particularly given the 10-week time constraint. As well, although I originally planned on storing pdf files as BLOBs (binary large objects) in MariaDB, I realised it'd be much more convenient (for both file management and database efficiency) to simply store the files in the localhost server directory.

From the ORM diagram, the optimal normal form algorithm was implemented to engineer nine redundancy-free normalised data tables. The overall database structure shown on the right illustrates the interrelationships between foreign keys.

NORMALISED TABLES

studentbase

snumber	lname	fname	ylevel	pc	house	email
varchar(10) MA PK	varchar(30) MA	varchar(30) MA	integer unsigned MA	varchar(2) MA	varchar(30) MA	varchar(50) MA

users

username	email	pwd	fname	lname	usertype	verified	token
varchar(30) MA PK	varchar(50) MA	varchar(40) MA	varchar(30) MA	varchar(30) MA	varchar(1) MA	boolean MA	varchar(40) OP

students

username	house	pc	ylevel	notes	permitted	formfilepath	resubmitpref
varchar(30) MA PK FK	varchar(30) MA	varchar(2) MA	integer unsigned MA	varchar(500) OP	boolean MA	varchar(100) MA	boolean MA

preferences

username	prefnum	week	day
varchar(30) MA PK FK	integer unsigned MA PK	varchar(1) MA	varchar(3) MA

events

eventid	eventdate	week	day	starttime	endtime	place	studentquota	organiserquota
integer unsigned auto_increment MA PK	date MA	integer unsigned MA	varchar(3) MA	time MA	time MA	varchar(80) MA	integer unsigned not null MA	integer unsigned not null MA

studentevents

username	eventid
varchar(30) MA PK FK	integer unsigned MA PK FK

reports

reportid	eventid	servings	feedback
integer unsigned auto_increment MA PK	integer unsigned MA FK	integer unsigned MA	varchar(500) OP

organiserevents

username	eventid
varchar(30) MA PK FK	integer unsigned MA PK FK

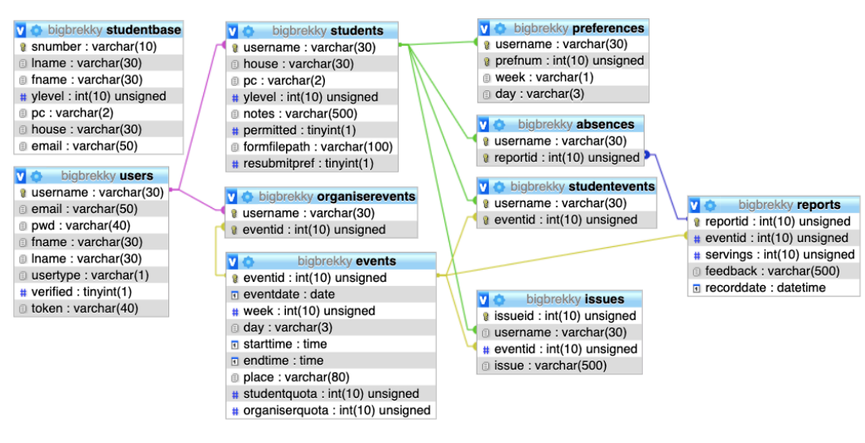
absences

username	reportid
varchar(30) MA PK FK	integer unsigned MA PK FK

issues

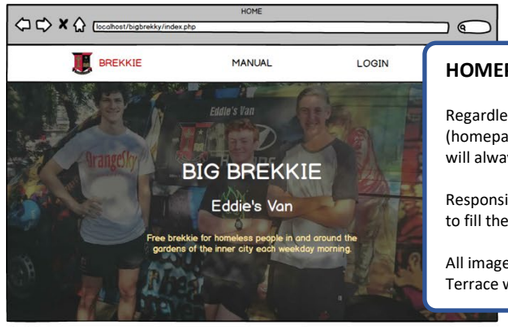
issueid	username	eventid	issue
integer unsigned auto_increment MA PK	varchar(30) OP FK	integer unsigned MA FK	varchar(500) MA

DATABASE STRUCTURE
(from phpMyAdmin)



2.5 USER INTERFACE WIREFRAMES & SITE MAP

GENERAL PAGES

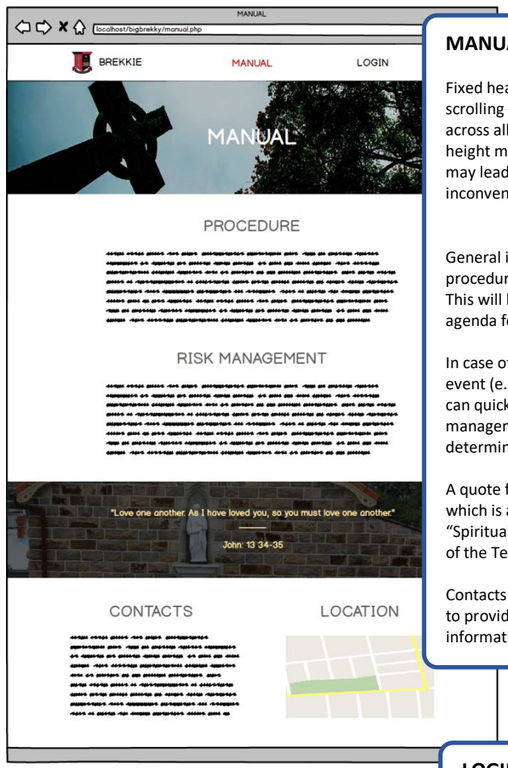


HOMEPAGE

Regardless of the usertype, 3 links (homepage, manual, login/register) will always be visible in the navbar.

Responsive – content will resize itself to fill the window screen size.

All images will be taken from the Terrace website.



MANUAL

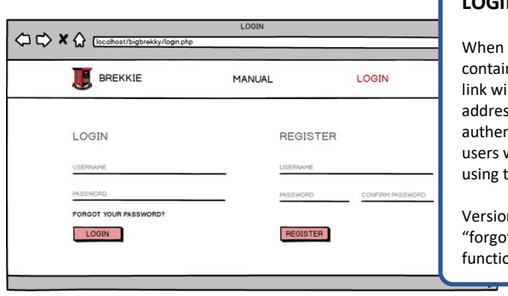
Fixed heading banner with parallel scrolling – consistently implemented across all webpages. The banner's height must not be too large as it may lead to unnecessary and inconvenient scrolling for the user.

General information describing the procedure for a Big Brekky event. This will be displayed as a timetable agenda for a typical morning session.

In case of an emergency during an event (e.g. fight breaks outs), users can quickly access these risk management instructions to determine an action plan.

A quote from the Gospel of John, which is also displayed under the "Spiritual Retreat Program" section of the Terrace website.

Contacts and location for Big Brekky to provide users with extensive information about this program.

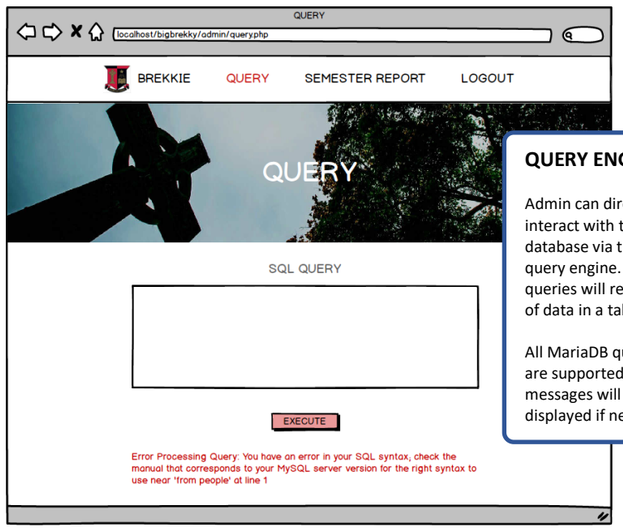


LOGIN/REGISTER

When users register, an email containing an account-verification link will be sent to their email address for 2-factor authentication. Once verified, users will then be able to login using their credentials.

Version 2.0: possibly implement "forgot your password" functionality.

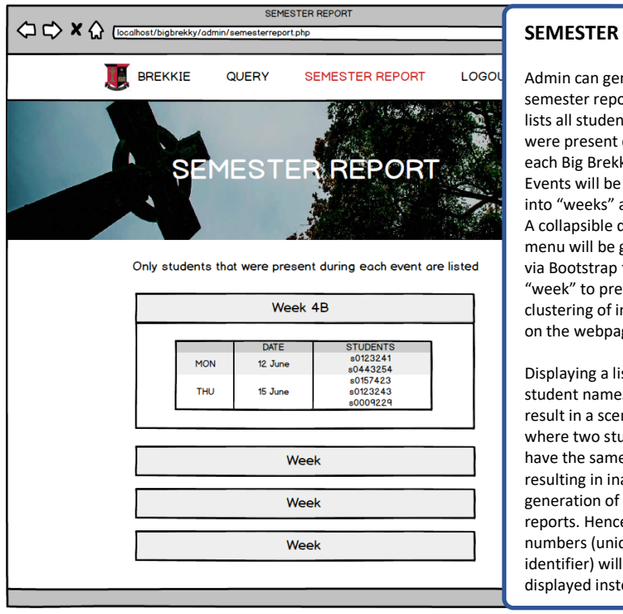
ADMIN



QUERY ENGINE

Admin can directly interact with the database via this SQL query engine. "SELECT" queries will return rows of data in a table format.

All MariaDB query types are supported and error messages will be displayed if necessary.

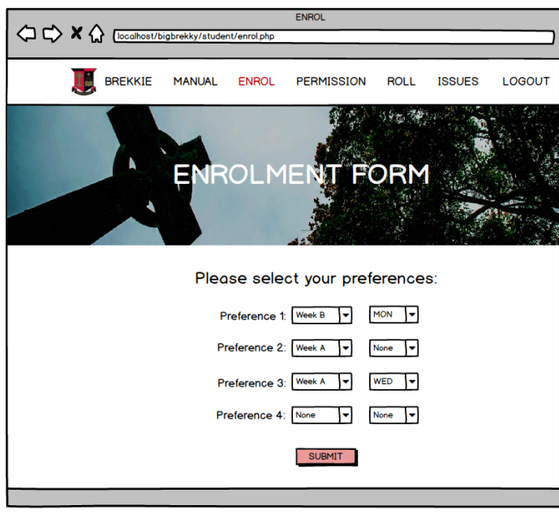


SEMESTER REPORT

Admin can generate a semester report, which lists all students that were present during each Big Brekky event. Events will be grouped into "weeks" as shown. A collapsible dropdown menu will be generated via Bootstrap for each "week" to prevent over-clustering of information on the webpage.

Displaying a list of student names may result in a scenario where two students have the same name, resulting in inaccurate generation of semester reports. Hence, s-numbers (unique identifier) will be displayed instead.

STUDENT

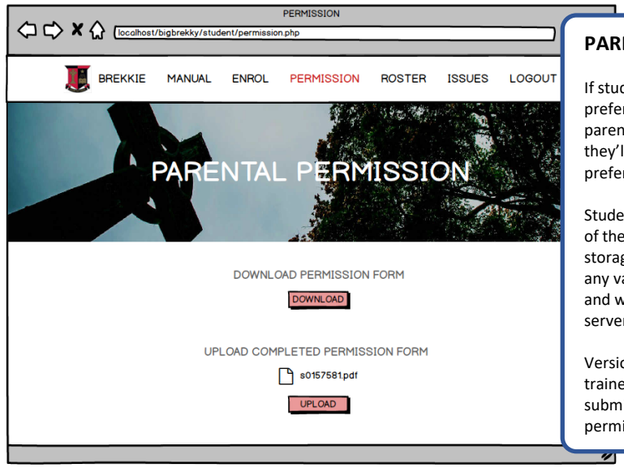


ENROLMENT FORM

Students can choose Big Brekky preferences consisting of week letter and weekday using a drop-down menu. Year 10 students can merely choose up to 2 preferences as their only options are Week A and B Fridays. Once submitted, this content will be replaced with a summary of their selected preferences and a form for requesting a resubmission.

Version 2.0: Students should also be allowed to submit specific dates on which they are not able to attend Big Brekky. This will prevent any last-minute roster changes.

Input fields will adjust their layout depending on screen size (bootstrap grid system).

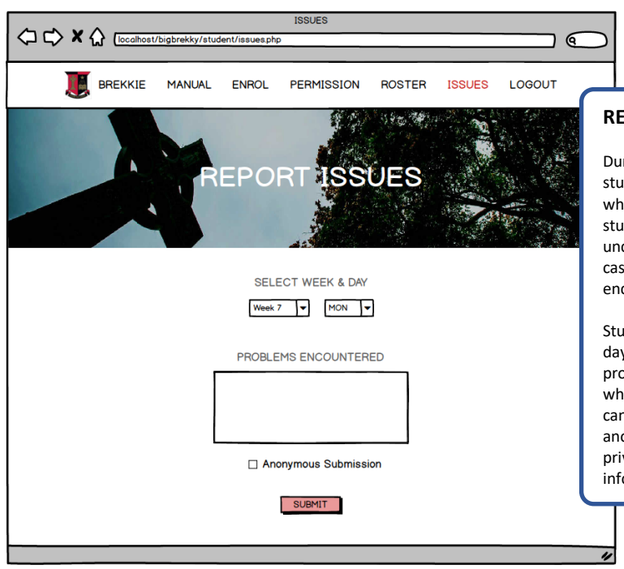


PARENTAL PERMISSION FORM

If students have submitted their preferences, they can download the parental permission slip. Otherwise, they'll be instructed to submit their preferences first.

Students can then upload a digital scan of their completed permission form for storage purposes. The file can be of any valid format (e.g. png, pdf, jpg) and will be saved inside a localhost server directory.

Version 2.0: an AI model could be trained to swiftly identify whether the submitted form is in fact, a parental permission form and completed.

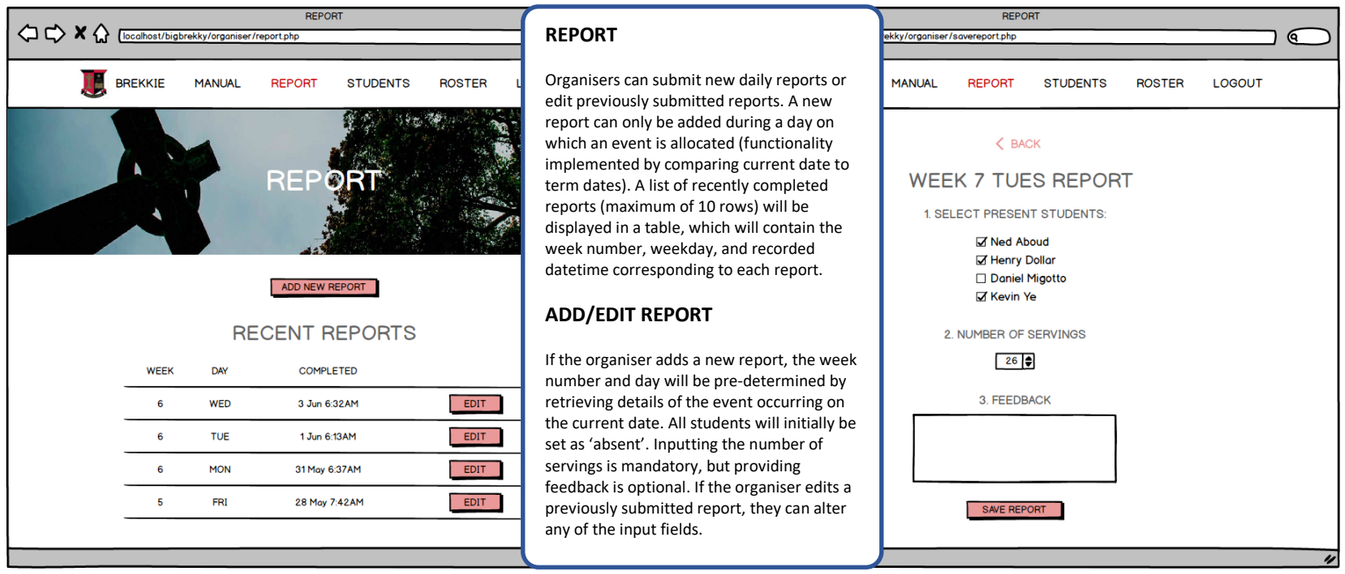


REPORT ISSUES

During a Big Brekky event, students may encounter a patron who may inflict distress upon the student or place him in an uncomfortable situation. In such cases, students should be encouraged to report issues.

Students can select the week and day they encountered the problem, and describe exactly what occurred. If need be, they can submit this form anonymously to ensure user privacy and safety of sensitive information.

ORGANISER

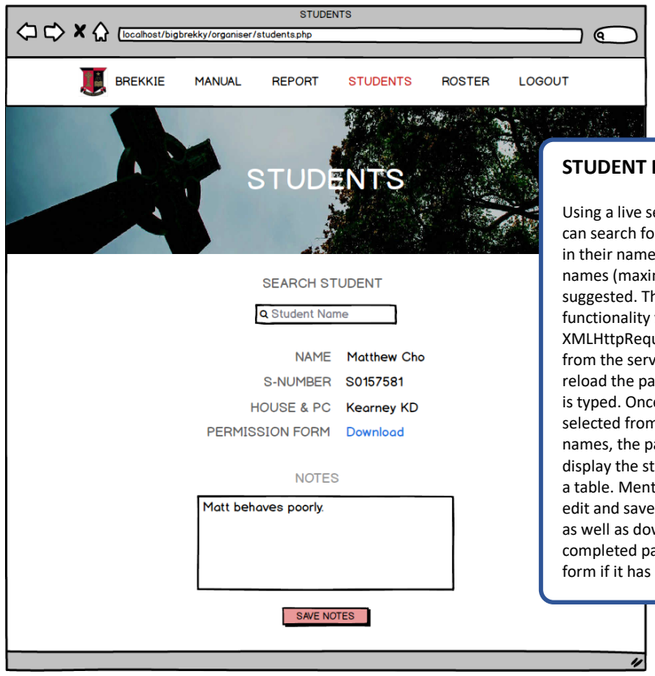


REPORT

Organisers can submit new daily reports or edit previously submitted reports. A new report can only be added during a day on which an event is allocated (functionality implemented by comparing current date to term dates). A list of recently completed reports (maximum of 10 rows) will be displayed in a table, which will contain the week number, weekday, and recorded datetime corresponding to each report.

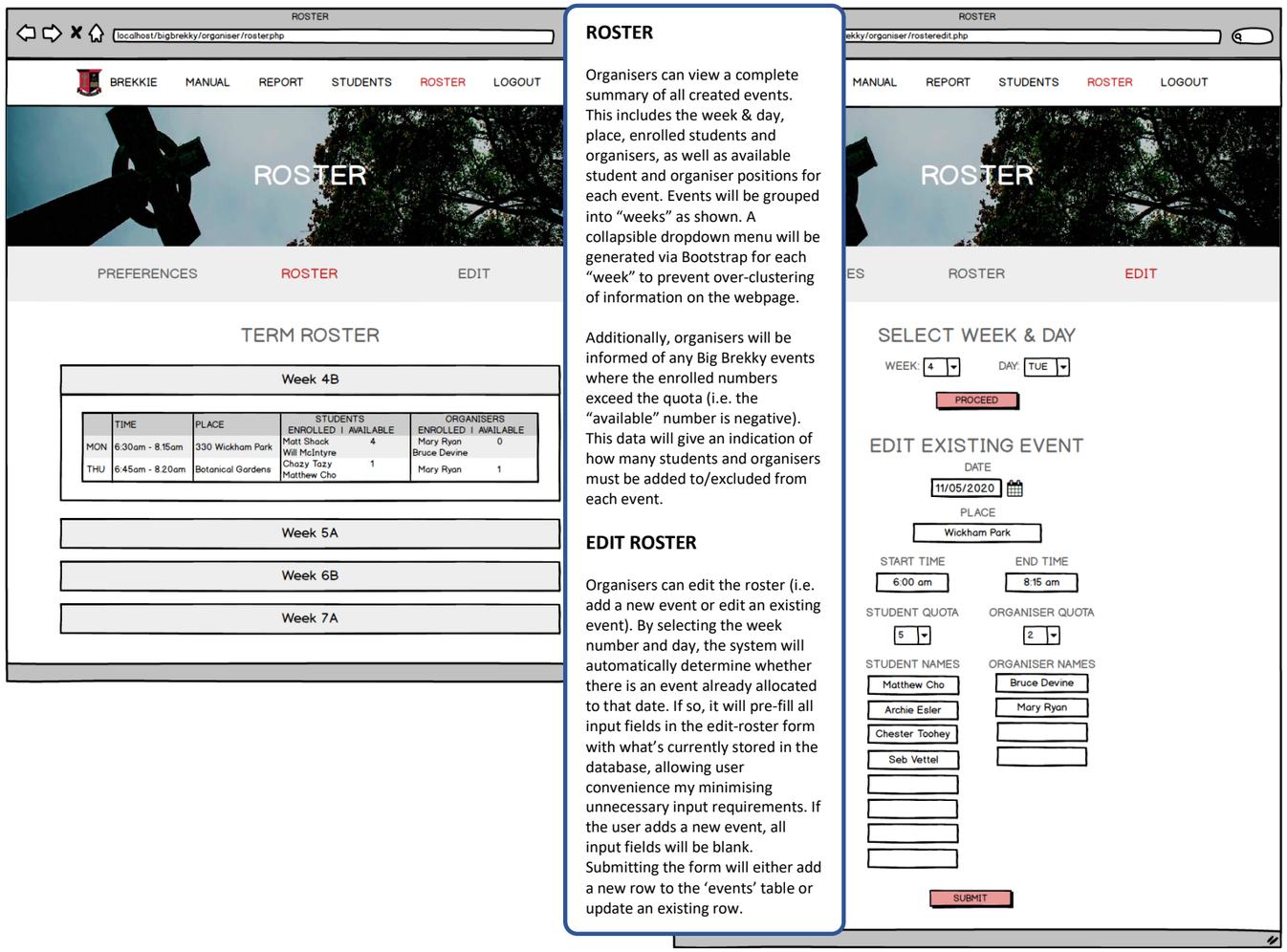
ADD/EDIT REPORT

If the organiser adds a new report, the week number and day will be pre-determined by retrieving details of the event occurring on the current date. All students will initially be set as 'absent'. Inputting the number of servings is mandatory, but providing feedback is optional. If the organiser edits a previously submitted report, they can alter any of the input fields.



STUDENT PROFILES

Using a live search bar, organisers can search for students by typing in their names. A list of possible names (maximum of 5) will be suggested. This live search functionality will be using XMLHttpRequest to request data from the server without having to reload the page each time a letter is typed. Once a student is selected from the list of suggested names, the page will retrieve and display the student's profile using a table. Mentors can subsequently edit and save student annotations, as well as download the student's completed parental permission form if it has been submitted.



ROSTER

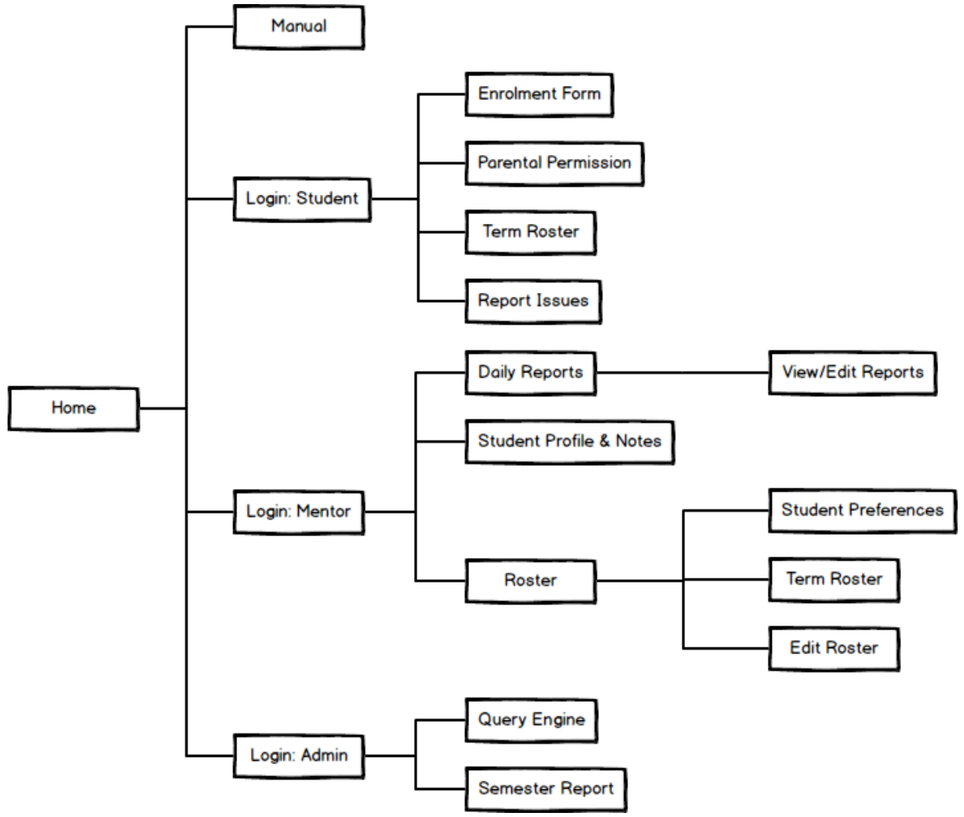
Organisers can view a complete summary of all created events. This includes the week & day, place, enrolled students and organisers, as well as available student and organiser positions for each event. Events will be grouped into "weeks" as shown. A collapsible dropdown menu will be generated via Bootstrap for each "week" to prevent over-clustering of information on the webpage.

Additionally, organisers will be informed of any Big Brekky events where the enrolled numbers exceed the quota (i.e. the "available" number is negative). This data will give an indication of how many students and organisers must be added to/excluded from each event.

EDIT ROSTER

Organisers can edit the roster (i.e. add a new event or edit an existing event). By selecting the week number and day, the system will automatically determine whether there is an event already allocated to that date. If so, it will pre-fill all input fields in the edit-roster form with what's currently stored in the database, allowing user convenience my minimising unnecessary input requirements. If the user adds a new event, all input fields will be blank. Submitting the form will either add a new row to the 'events' table or update an existing row.

SITE MAP



3.0 DEVELOPMENT

3.1 DATABASE TABLES & SQL COMMANDS (CREATE TABLE & INSERT INTO)

STUDENTBASE

The studentbase table is an external dataset (i.e. a CSV file was imported into phpMyAdmin). It will be provided and managed by the admin. It'll contain data about students in Years 10 to 12 and will be updated annually.

snumber	lname	fname	ylevel	pc	house	email
S0072852	Howard	Michael	12	BF	Barrett	S0072852@terrace.qld.edu.au
S0077792	Soong	Adam	12	WG	Windsor	S0077792@terrace.qld.edu.au
S0086472	Toohey	Chester	12	ME	Mahoney	S0086472@terrace.qld.edu.au
S0090222	McIntyre	Will	12	KA	Kearney	S0090222@terrace.qld.edu.au

```
CREATE TABLE studentbase (
    snumber varchar(10) not null,
    lname varchar(30) not null,
    fname varchar(30) not null,
    ylevel integer unsigned not null,
    pc varchar(2) not null,
    house varchar(30) not null,
    email varchar(50) not null,
    primary key(snumber));

INSERTT INTO studentbase
VALUES ('S0072852', 'Howard', 'Michael', 12, 'BF', 'Barrett',
'S0072852'@terrace.qld.edu.au');
```

EVENTS

The events table simply contains data about each created event. They are uniquely identified by 'eventid'. When editing/adding events, it is assumed that no more than one event can occur during a day.

eventid	eventdate	week	day	starttime	endtime	place	studentquota	organiserquota
1	2020-05-18	5	mon	06:30:00	08:20:00	330 Wickham Terrace, Spring Hill	4	1
2	2020-06-03	7	wed	05:30:00	08:10:00	454 Gregory Terrace, Spring Hill	8	2
3	2020-05-28	6	thu	06:30:00	07:30:00	330 Wickham Terrace, Spring Hill	6	3
4	2020-05-20	5	wed	05:15:00	08:00:00	454 Gregory Terrace, Spring Hill	6	2

```
CREATE TABLE events (
    eventid integer unsigned auto_increment not null,
    eventdate date not null,
    week integer unsigned not null,
    day varchar(3) not null,
    starttime time not null,
    endtime time not null,
    place varchar(80) not null,
    studentquota integer unsigned not null,
    organiserquota integer unsigned not null,
    primary key(eventid));

INSERTT INTO events
VALUES ('2020-05-18', 5, 'mon', '06:30:00', '08:20:00', '330
Wickham Terrace, Spring Hill', 4, 1);
```

USERS

The users table will contain data about users that have been registered. The 'token' field is transient data – once the user activates his/her account, the 'token' and 'activated' fields will be updated to Null and 1 respectively.

username	email	pwd	fname	lname	usertype	verified	token
admin	bigbrekkyadmin@terrace.qld.edu.au	d033e22ae348aeb5660fc2140aec35850c4da997	admin	admin	a	1	*NULL*
bruce devine	BruceDevine@terrace.qld.edu.au	e3f284cae2097a55ddbaf92f45b91ea8506b49bf	Bruce	Devine	o	1	*NULL*
mary ryan	MaryRyan@terrace.qld.edu.au	8e7be411ad89ade93d144531f3925d0bb4011004	Mary	Ryan	o	0	7b52009b64fd0a2a49e6d8a939753077792b0554
peter whitehouse	PeterWhitehouse@terrace.qld.edu.au	40bd001563085fc35165329ea1ff5c5ecbdbbeef	Peter	Whitehouse	o	1	*NULL*
s0086472	S0086472@terrace.qld.edu.au	356a192b7913b04c54574d18c28d46e6395428ab	Chester	Toohey	s	0	0ec09ef9836da03f1add21e3ef607627e687e790
s0090222	S0090222@terrace.qld.edu.au	af06318c33c8e41c70083ee23dbe19426f1f9c5b	Will	McIntyre	s	1	*NULL*

```
CREATE TABLE users (
    username varchar(30) not null,
    email varchar(50) not null,
    pwd varchar(40) not null,
    fname varchar(30) not null,
    lname varchar(30) not null,
    usertype varchar(1) not null,
    verified boolean not null,
    token varchar(40),
    primary key(username));

INSERT INTO users
VALUES ('mary ryan', 'MaryRyan@terrace.qld.edu.au',
'8e7be411ad89ade93d144531f3925d0bb4011004', 'Mary', 'Ryan', 'o', 0,
'7b52009b64fd0a2a49e6d8a939753077792b0554');
```

STUDENTEVENTS & ORGANISEREVENTS

Studentevents (left) and organiserevents (right) tables show which students and organisers are allocated to which events, having the same two fields: username and eventid. However, there is one slight difference: the usernames for studentevents are s-numbers, whereas the usernames for organiserevents are first and last names.

username	eventid
s0086472	1
s0086472	2
s0086472	3
s0086472	4

username	eventid
bruce devine	2
bruce devine	5
mary ryan	2
mary ryan	3

```
CREATE TABLE studentevents (
    username varchar(30) not null,
    eventid integer unsigned not null,
    primary key(username, eventid),
    foreign key(username) references students(username),
    foreign key(eventid) references events(eventid));

INSERTT INTO studentevents
VALUES ('s0086472', 1);
```

STUDENTS

The 'resubmitpref' field indicates whether or not the student has requested for resubmission of his Big Brekky preferences.

username	house	pc	ylevel	notes	permitted	formfilepath	resubmitpref
s0086472	Mahoney	ME	12	*NULL*	0	permissionforms/s0086472.pdf	0
s0090222	Kearney	KA	12	Will was very generous.	1	permissionforms/s0090222.pdf	0
s0095712	Treacy	TC	12	*NULL*	0	permissionforms/s0095712.pdf	0
s0100491	Reidy	RG	12	Ben has been good	1	permissionforms/s0100491.pdf	1

```
CREATE TABLE students (
    username varchar(30) not null,
    house varchar(30) not null,
    pc varchar(2) not null,
    ylevel integer unsigned not null,
    annotations varchar(500),
    permitted boolean not null,
    formfilepath varchar(100) not null,
    resubmitpref boolean not null,
    primary key(username),
    foreign key(username) references users(username));

INSERT INTO students (username, house, pc, ylevel,
permitted, formfilepath, resubmitpref)
VALUES ('s0086472', 'Mahoney', 'ME', 12, 0,
'permissionforms/s0086472.pdf', 0);
```

ISSUES

The issues table stores any issues that students encountered during a Big Brekky session (submitted by the impacted student). The 'username' field can be NULL if the student requests for an anonymous submission.

issueid	username	eventid	issue
1	*NULL*	1	A patron approached me and offered drugs
2	s0090222	5	I felt uncomfortable throughout the entire session
3	s0157581	1	I was kicked by a man
4	*NULL*	3	Listening to a patron's life story gave me anxiety

```
CREATE TABLE issues (
    issueid integer unsigned auto_increment not null,
    username varchar(30),
    eventid integer unsigned not null,
    issue varchar(500) not null,
    primary key(issueid),
    foreign key(username) references students(username),
    foreign key(eventid) references events(eventid));

INSERTT INTO issues (eventid, issue)
VALUES (1, 'A patron approached me and offered drugs');
```

PREFERENCES

The 'prefnum' field contains a number from 1 to 5 as non-Year 10 students can select up to 5 preferences (Year 10 students can select 2).

username	prefnum	week	day
s0086472	1	a	mon
s0086472	2	a	thu
s0086472	3	b	fri
s0157581	1	a	mon

```
CREATE TABLE preferences (
    username varchar(30) not null,
    prefnum integer unsigned not null,
    week varchar(1) not null,
    day varchar(3) not null,
    primary key(username, prefnum),
    foreign key(username) references students(username));

INSERT INTO preferences
VALUES ('s0086472', 1, 'a', 'mon');
```

REPORTS

The 'eventid' field indicates the event during which each report was completed. The 'recorddate' field includes the time when the roll was recorded (datetime value) to disincentivise organisers from delaying their report submissions.

reportid	eventid	servings	feedback	recorddate
1	1	26	Awesome job from the boys.	2020-05-18 07:49:10
2	3	11	Rained a lot today.	2020-05-28 08:10:23
4	2	42	Lovely weather today!	2020-06-03 14:46:13
5	4	17	*NULL*	2020-05-20 07:13:00

```
CREATE TABLE reports (
    reportid integer unsigned auto_increment not null,
    eventid integer unsigned not null,
    servings integer unsigned not null,
    feedback varchar(500),
    primary key(reportid),
    foreign key(eventid) references events(eventid));

INSERTT INTO reports
VALUES (1, 26, 'Awesome job from the boys', '2020-05-18
07:49:10');
```

ABSENCES

The absences table records student 'username' and the 'reportid' of the reports he were absent for.

username	reportid
s0090222	4
s0095712	1
s0100491	1
s0103941	4

```
CREATE TABLE absences (
    username varchar(30) not null,
    reportid integer unsigned not null,
    primary key(username, reportid),
    foreign key(username) references students(username),
    foreign key(reportid) references dailyreports(reportid));

INSERTT INTO absences
VALUES ('s0090222', 4);
```

3.2 EXAMPLE QUERIES

Updating event details
(MeekroDB library syntax)

```
DB::update('events', array(
    'eventdate' => $_POST['eventdate'],
    'week' => $_POST['week'],
    'day' => $_POST['day'],
    'starttime' => $_POST['starttime'],
    'endtime' => $_POST['endtime'],
    'place' => $_POST['place'],
    'studentquota' => $_POST['studentquota'],
    'organiserquota' => $_POST['organiserquota']
), 'eventid = %s', $eventid);
```

Retrieve names of students and organisers allocated to events in the specified week, grouped by rows of events - student and organiser names are concatenated in their respective columns

```
select e.*,
    group_concat(distinct concat(upper(susers.lname), " ", susers.fname) order by susers.lname) as students,
    group_concat(distinct concat(upper(ousers.lname), " ", ousers.fname) order by ousers.lname) as organisers
from events e
    left join studentevents s on e.eventid = s.eventid
    left join users susers on s.username = susers.username
    left join organiserevents o on e.eventid = o.eventid
    left join users ousers on o.username = ousers.username
where e.week = $weeknum
group by e.eventid'
```

Using a subquery to retrieve students attending the specified event

```
select username, concat(fname, " ", lname) as name
from users
where username in (select username
                   from studentevents
                   where eventid = $eventid)
order by lname
```

3.3 ALGORITHMS OF MAJOR FUNCTIONALITIES

USER REGISTRATION

Registration

```
BEGIN
IF registration attempted THEN
     INPUT username [from POST] – sanitised
     INPUT pwd [from POST] – sha1 hashed
     INPUT confirmpwd [from POST] – sha1 hashed
     CONNECT to database
     INIT usersData = SELECT user's general details from users table
     INIT studentbaseData = SELECT user's student details from studentbase table
     IF usersData exists THEN
          OUTPUT 'username already registered'
     ELSE IF pwd and confirmpwd match THEN
          INIT token = sha1 hashed random integer from 0 to 1000
          IF studentbaseData doesn't exist THEN
               INIT splitname = split username into an array of strings with delimeter ' '
               IF splitname array doesn't have two elements THEN
                    OUTPUT 'invalid organiser username – please enter in first and last names separated
                    by a space'
               ELSE
                    INIT firstname = splitusername[0]
                    INIT lastname = splitusername[1]
                    INIT email = firstname + lastname + '@terrace.qld.edu.au'
                    INSERT organiser's general user data into users table (
                         'username' => username,
                         'email' => email,
                         'pwd' => pwd,
                         'fname' => firstname,
                         'lname' => lastname,
                         'usertype' => 'o',
                         'verified' => 0,
                         'token' => token)
                    send account activation email (link has URL variables username and token)
               ENDIF
          ELSE
               INIT email = email in studentbaseData array
               INIT firstname = fname in studentbaseData array
               INIT lastname = lname in studentbaseData array
               INSERT student's general user data into users table (
                    'username' => username,
                    'email' => email,
                    'pwd' => pwd,
                    'fname' => firstname,
                    'lname' => lastname,
                    'usertype' => 's',
                    'verified' => 0,
                    'token' => token)
               INIT house = house in studentbaseData array
               INIT pc = pc in studentbaseData array
               INIT ylevel = ylevel in studentbaseData array
               INIT formfilepath = 'permissionforms/' + username + '.pdf'
               INSERT user's student data into students table (
                    'username' => username,
                    'house' => house,
                    'pc' => pc,
                    'ylevel' => ylevel,
                    'permitted' => 0,
                    'formfilepath' => formfilepath,
                    'resubmitpref' => 0)
               send account verification email (link has URL variables username and token)
          ENDIF
     ELSE
          OUTPUT 'passwords do not match'
     END IF
END IF
END
```

Account Activation

```
BEGIN
IF username and token [from GET] are set (user has clicked the account activation button via email) THEN
     CONNECT to database
     usersData = SELECT user's general details from users table
     IF user's account is already activated THEN
          OUTPUT 'your account has already been activated'
     ELSE
          IF username and token [from GET] match correct values in usersData THEN
               UPDATE activation status in users table (
                    'token' -> NULL,
                    'activated' -> 1)
               OUTPUT 'account has been activated – please proceed to login'
          END IF
     END IF
END IF
END
```

USER LOGIN

```
BEGIN
IF login attempted THEN
     INPUT username [from POST] – sanitised
     INPUT password [from POST] – sha1 hashed
     CONNECT to database
     usersData = SELECT user's general details from users table
     IF usersData doesn't exist THEN
          OUTPUT 'invalid email or password'
     ELSE
          IF password is incorrect THEN
               OUTPUT 'invalid email or password'
          ELSE
               IF user's account is activated THEN
                    OUTPUT 'log in successful'
                    username [SESSION] = username in usersData array
               ELSE
                    OUTPUT 'activated your account via email to login'
          END IF
     END IF
END IF
END
```

STUDENT: PREFERENCE SELECTION

Processing Preferences

```
BEGIN
INIT userPrefData = SELECT user's Big Brekky preferences from preferences table
IF preferences form submitted and userPrefData doesn't exist THEN
     INPUT pref1 [from POST]
     INPUT pref2 [from POST]
     IF student is not in Year 10 THEN
          INPUT pref3 [from POST]
          INPUT pref4 [from POST]
          INPUT pref5 [from POST]
     ELSE
          INIT pref3 = pref4 = pref5 = null;
     ENDIF

     INIT prefArray = [pref1, pref2, pref3, pref4, pref5]
     FOREACH pref IN prefArray
          IF pref is null THEN
               delete pref from prefArray
          ENDIF
     ENDFOR
     IF number of preferences in prefArray is greater than 0 THEN
          IF student has selected duplicate weekdays THEN
               OUTPUT 'you cannot select duplicate weekdays'
          ELSE
               FOREACH pref IN prefArray
                    INSERT student's preferences into preferences table (
                         'username' => student's username,
                         'prefnum' => CALCULATE  (index of pref in prefArray) + 1,
                         'week' => week of pref,
                         'day' => day of pref)
               ENDFOR
               OUTPUT 'please download the parental permission form'
          ENDIF
     ELSE
          OUTPUT 'please select a weekday'
     ENDIF
ENDIF
END
```

Resubmission Request

```
BEGIN
IF student requested to resubmit preferences THEN
     INPUT reason [from POST]
     send notification email to admin (contains reason for requesting resubmission)
     OUTPUT 'your request has been sent'
     UPDATE resubmission status in students table ('resubmitpref' => 1)
ENDIF
END
```

ORGANISER: SEARCH & VIEW STUDENT PROFILE

AJAX XMLHttpRequest For Live Name Search

```
BEGIN
IF search bar text changes THEN
     INPUT studentName [retrieve search bar text using jQuery]
     IF studentName length is 0 THEN
          clear suggestions
     ELSE
          INIT request = new XMLHttpRequest()
          FUNCTION request.onreadystatechange()
               //called by AJAX whenever readyState property of server  request is changed
               IF request has finished and server response is ready THEN
                    OUTPUT student name suggestions
               ENDIF
          ENDFUNCTION
          send server request to 'namesuggestions.php?name=' + studentName
     ENDIF
ENDIF
END
```

Getting Student Name Suggestions (namesuggestions.php)

```
BEGIN
GET name [from REQUEST]
INIT suggestedNames = ''
IF name is empty string THEN
     CONNECT to database
     INIT possibleStudents = SELECT max of 5 students from users table where full name is like name
     FOREACH student IN possibleStudents
          suggestedNames = suggestedNames + (student username as GET variable)
     END FOR
END IF
IF suggestedNames is empty string THEN
     OUTPUT 'no suggestion'
ELSE
     OUTPUT suggestedNames
END IF
END
```

Displaying Student Profile

```
BEGIN
IF username [from GET] exists in URL THEN
     usersProfile = SELECT user details from users table
     IF usersProfile exists THEN
          studentsProfile = SELECT student details from students table
          OUTPUT studentsProfile (name, snumber, house & PC group) organised into a table
          IF student has submitted a parental permission form THEN
               OUTPUT 'download student's permission form'
          ELSE
               OUTPUT 'permission form not submitted'
          ENDIF
          OUTPUT student notes from studentsProfile array
     ENDIF
ENDIF
END
```

ORGANISER: EDIT ROSTER

```
BEGIN
IF organiser has submitted the edit roster form THEN
     IF organiser has added a new event THEN
          INSERT newly added event details into events table (
               'eventdate' => INPUT eventdate [from POST],
               'week' => INPUT week [from POST],
               'day' => INPUT day [from POST],
               'starttime' => INPUT starttime [from POST],
               'endtime' => INPUT endtime [from POST],
               'place' => INPUT place [from POST],
               'studentquota' => INPUT studentquota [from POST],
               'organiserquota' => INPUT organiserquota [from POST])
          INIT eventid = newly inserted events's eventid from events table
     ELSE
          //organiser has edited an existing event
          INPUT eventid [from POST]
          UPDATE edited event details in events table (
               'eventdate' => INPUT eventdate [from POST],
               'week' => INPUT week [from POST],
               'day' => INPUT day [from POST],
               'starttime' => INPUT starttime [from POST],
               'endtime' => INPUT endtime [from POST],
               'place' => INPUT place [from POST],
               'studentquota' => INPUT studentquota [from POST],
               'organiserquota' => INPUT organiserquota [from POST])
     ENDIF
     DELETE all students previously allocated to this event in studentevents table
     INIT studentnames = an array of 8 newly allocated students' names [from POST]
     FOREACH student IN studentnames
          INIT username = SELECT username of student from users table
          IF username exists THEN
               INSERT newly allocated student to studentevents table (
                    'username' => username,
                    'eventid' => eventid)
          ENDIF
     ENDFOR

     DELETE all organisers previously allocated to this event in organiserevents table
     INIT organisernames = an array of 4 newly allocated organisers' names [from POST]
     FOREACH organiser IN organisernames
          INIT username = SELECT username of organiser from users table
          IF username exists THEN
               INSERT newly allocated organiser to organiserevents table (
                    'username' => username,
                    'eventid' => eventid)
          ENDIF
     ENDFOR
     OUTPUT 'roster has been saved'
ENDIF
END
```

ORGANISER: EDIT/ADD NEW REPORT

```
BEGIN
IF organiser saves report THEN
     INPUT reportid [from POST]
     INPUT eventid [from POST]
     INPUT servings [from POST]
     INPUT feedback [from POST]
     INIT recordDate = current date in PHP date format (Y-m-d H:i:s)
     IF presentStudents [from POST] exists THEN
          INIT presentStudents = presentStudents [from POST]
     ELSE
          INIT presentStudents = []
     ENDIF
     INIT eventStudents = SELECT students attending specified event from studentevents table
     INIT absentStudents = students in eventStudents that are not in presentStudents
     IF reportid is NULL THEN
          // new report created
          INSERT into reports table (
               'eventid => eventid,
               'servings' => servings,
               'feedback' => feedback,
               'recorddate' => recordDate)
          INIT newreportid = reportid of newly inserted report in reports table
          FOREACH student IN absentStudents
               INSERT into absences table (
                    'username' => student,
                    'reportid' => newreportid)
          ENDFOR
     ELSE
          // existing report edited
          UPDATE servings and feedback of selected report in reports table
          DELETE all absent students for selected report in absences table
          FOREACH student IN absentStudents
               INSERT into absences table (
                    'username' => student,
                    'reportid' => reportid)
          ENDFOR
     ENDIF
     OUTPUT 'roll has been saved'
ENDIF
END
```

4.0 CODE

Only the most important code snippets have been included

4.1 Login/Registration

```
<?php
if (isset($_POST['submitlogin'])) {
     //log in attempted

     //get submitted credentials and sanitise/hash them
     $username =
strtolower(trim(stripslashes(htmlspecialchars($_POST['username']))));
     $pwd = sha1($_POST['pwd']);

     //connect to database
     connectDB();

     //attempt to get user's data
     $userData = DB::queryFirstRow('select * from users where username = %s',
$username);
     if (!$userData) {
          //input username not found in database
          alert('danger', 'INVALID USERNAME OR PASSWORD');
     } else {
          //live user - check password
          if ($pwd != $userData['pwd']) {
               //invalid password
               alert('danger', 'INVALID USERNAME OR PASSWORD');
          } else {
               //user credentials match
               if ($userData['verified']) {
                    //if 2FA successful - account verified
                    //set session variables for checking login state
                    $_SESSION['usertype'] = $userData['usertype'];
                    $_SESSION['username'] = $userData['username'];
                    $_SESSION['fname'] = $userData['fname'];
                    $_SESSION['loginSuccessful'] = true;

                    //redirect to previous page
                    header('Location: '.$_SESSION['prevPage']);
                    exit();
               } else {
                    alert('danger', 'PLEASE VERIFY YOUR ACCOUNT VIA EMAIL TO
LOGIN');
               }
          }
     }

} elseif (isset($_POST['submitregister'])) {
     //sign up attempted
     //get submitted credentials and sanitise/hash them
     $username =
strtolower(trim(stripslashes(htmlspecialchars($_POST['username']))));
     $pwd = sha1($_POST['pwd']);
     $confirmpwd = sha1($_POST['confirmpwd']);

     //connect to dabatase
     connectDB();

     //to check if username already exists
     $usersData = DB::queryFirstRow('select * from users where username = %s',
$username);
     //to check if username is a registrable student
     $studentbaseData = DB::queryFirstRow('select * from studentbase where
snumber = %s', $username);

     if ($usersData) {
          //username already exists
          alert('danger', 'USERNAME ALREADY REGISTERED');
     } elseif ($pwd == $confirmpwd) {
          //passwords match

          //generating url token for account verification
          $token = sha1(rand(0, 1000));

          if (!$studentbaseData) {
               //if the registering user is not a student, new organiser user
               //split the username into first and last names
               $splitusername = explode(" ", $username);
               if (count($splitusername) != 2) {
                    //organiser username is not valid
                    alert('danger', 'INVALID ORGANISER USERNAME - ENTER YOUR
FIRST AND LAST NAMES SEPARATED BY A SPACE');
               } else {
                    //valid organiser username
                    //extract first and last names and capitalise first
letter
                    $fname = ucwords($splitusername[0]);
                    $lname = ucwords($splitusername[1]);
                    $email = $fname.$lname."@terrace.qld.edu.au";

                    //organiser registration - add to users table
                    DB::insert('users', array(
                         'username' => $username,
                         'email' => $email,
                         'pwd' => $pwd,
                         'fname' => $fname,
                         'lname' => $lname,
                         'usertype' => 'o',
                         'verified' => 0,
                         'token' => $token
                    ));

                    //set session variables for checking login state
                    $_SESSION['signupSuccessful'] = true;

                    //send account-verification username
                    require('sendmail.php');
                    sendVerificationEmail($email, $username, $token);

                    //redirect to previous page
                    header('Location: '.$_SESSION['prevPage']);
                    exit();
               }
          } else {
               //new student user
               $email = $studentbaseData['email'];
               $fname = $studentbaseData['fname'];
               $lname = $studentbaseData['lname'];

               //student registration - add to users table
               DB::insert('users', array(
                    'username' => $username,
                    'email' => $email,
                    'pwd' => $pwd,
                    'fname' => $fname,
                    'lname' => $lname,
                    'usertype' => 's',
                    'verified' => 0,
                    'token' => $token
               ));

               $house = $studentbaseData['house'];
               $pc = $studentbaseData['pc'];
               $ylevel = $studentbaseData['ylevel'];
               $formfilepath = 'permissionforms/'.$username.'.pdf';

               //add student to students table
               DB::insert('students', array(
                    'username' => $username,
                    'house' => $house,
                    'pc' => $pc,
                    'ylevel' => $ylevel,
                    'permitted' => 0,
                    'formfilepath' => $formfilepath,
                    'resubmitpref' => 0
               ));

               //set session variables for checking login state
               $_SESSION['signupSuccessful'] = true;

               //send account-verification username
               require('sendmail.php');
               sendVerificationEmail($email, $username, $token);

               //redirect to previous page
               header('Location: '.$_SESSION['prevPage']);
               exit();
          }
     } else {
          //passwords do not match
          alert('danger', 'PASSWORDS DO NOT MATCH');
     }
}
?>
```

4.2 Preference Selection

```
<?php
//retrieving user's preferences data
$userPrefData = DB::query('select day from preferences where username = %s
order by prefnum', $_SESSION['username']);

//student year level
$studentYear = DB::queryFirstRow('select ylevel from students where username
= %s', $_SESSION['username'])['ylevel'];

//preferences form submitted & user has not already submitted preferences
if (isset($_POST['submitpref']) && !$userPrefData) {
     $pref1 = $_POST['pref1week'].' '.$_POST['pref1day'];
     $pref2 = $_POST['pref2week'].' '.$_POST['pref2day'];
     if ($studentYear != 10) {
          //student not in year 10 can choose up to 5 preferences
          $pref3 = $_POST['pref3week'].' '.$_POST['pref3day'];
          $pref4 = $_POST['pref4week'].' '.$_POST['pref4day'];
          $pref5 = $_POST['pref5week'].' '.$_POST['pref5day'];
     } else {
          //student in year 10 can only choose up to 2 preferences
          $pref3 = $pref4 = $pref5 = 'none none';
     }

     //discard preferences where the student has selected none for week or day
     $prefArray = [$pref1, $pref2, $pref3, $pref4, $pref5];
     for ($i=0; $i<5; $i++) {
          //split each preference into week and day
          $pref = explode(' ', $prefArray[$i]);
          //if the week or day is 'none', discard that preference
          if ($pref[0] == 'none' || $pref[1] == 'none') {
               unset($prefArray[$i]);
          }
     }
     //re-indexing the array
     $prefArray = array_values($prefArray);

     if (count($prefArray) != 0) {
          //student has selected at least 1 preference
          if (count($prefArray) !== count(array_unique($prefArray))) {
               //duplicate preferences selected
               alert('danger', 'YOU CANNOT SELECT DUPLICATE WEEKDAYS');
          } else {
               for ($i=0; $i<count($prefArray); $i++) {
                    //split preference into week and day
                    $pref = explode(' ', $prefArray[$i]);
                    //add each preference to preferences table
                    DB::insert('preferences', array(
                         'username' => $_SESSION['username'],
                         'prefnum' => $i+1,
                         'week' => $pref[0],
                         'day' => $pref[1]
                    ));
               }

               alert('info', 'PLEASE DOWNLOAD THE PARENTAL PERMISSION FORM');
          }
     } else {
          //student has not selected any preferences
          alert('danger', 'PLEASE SELECT A WEEKDAY');
     }
}

//for checking if user has submitted the preference-resubmission-request form
$userResubmitpref = DB::queryFirstRow('select resubmitpref from students where
username = %s', $_SESSION['username'])['resubmitpref'];
//resubmission-request form submitted & user has not already requested
if (isset($_POST['submitrequest']) && !$userResubmitpref) {
     //student's provided reason for resubmission request
     $reason = $_POST['resubmitreason'];

     //send admin an email (my email address used for testing)
     require('../sendmail.php');
     sendResubmitprefEmail($_SESSION['username'], $reason);

     alert('info', 'YOUR REQUEST HAS BEEN SENT');

     //update students' resubmitpref status to true
     DB::update('students', array(
          'resubmitpref' => 1,
     ), 'username = %s', $_SESSION['username']);
}
?>

<?php
//retrieving user's preferences data
$userPrefData = DB::query('select * from preferences where username = %s order
by prefnum', $_SESSION['username']);

if (!$userPrefData) {
     //student has not already submitted preferences form
     echo '
     <form name="prefform" method="post" action="enrol.php" class="enrol-form
form-horizontal">
          <h2>Preference Selection</h2>';

     if ($studentYear != 10) {
          //student is not in Year 10 - cannot choose Fridays
          echo '<p>Fridays are only available to Year 10 students.</p>';
          for ($i=1; $i<=5; $i++) {
               echo '
               <div class="form-group form-select-group">
                    <label class="control-label" for="pref'.$i.'">Preference
'.$i.'</label>
                    <select name="pref'.$i.'week" class="pref-select"
id="pref'.$i.'week">
                         <option value="none" selected>NONE</option>
                         <option value="a">WEEK A</option>
                         <option value="b">WEEK B</option>
                    </select>
                    <select name="pref'.$i.'day" class="pref-select"
id="pref'.$i.'day">
                         <option value="none" selected>NONE</option>
                         <option value="mon">MON</option>
                         <option value="tue">TUE</option>
                         <option value="wed">WED</option>
                         <option value="thu">THU</option>
                    </select>
               </div>';
          }
     } else {
          //student is in Year 10 - can only choose Fridays
          echo '<p>Year 10 students can only choose Fridays.</p>';
          for ($i=1; $i<=2; $i++) {
               echo '
               <div class="form-group form-select-group">
                    <label class="control-label" for="pref'.$i.'">Preference
'.$i.'</label>
                    <select name="pref'.$i.'week" class="pref-select"
id="pref'.$i.'week">
                         <option value="none" selected>NONE</option>
                         <option value="a">WEEK A</option>
                         <option value="b">WEEK B</option>
                    </select>
                    <select name="pref'.$i.'day" class="pref-select"
id="pref'.$i.'day">
                         <option value="none" selected>NONE</option>
                         <option value="fri">FRI</option>
                    </select>
               </div>';
          }
     }

     echo '
          <input class="button" type="submit" name="submitpref"
value="SUBMIT">
     </form>';
} else {
     //student has already submitted preferences form

     //displaying student preferences
     echo '<h2>Your Selected Preferences</h2>';
     for ($i=0; $i<count($userPrefData); $i++) {
          $prefweek = strtoupper($userPrefData[$i]['week']);
          $prefday = strtoupper($userPrefData[$i]['day']);
          echo '<h3>Preference '.($i+1).': Week '.$prefweek.'
'.$prefday.'</h3>';
     }

     echo '<h2>Request For Resubmission</h2>';

     //checking if user has already requested for a resubmission
     $resubmitpref = DB::queryFirstRow('select resubmitpref from students
where username = %s', $_SESSION['username'])['resubmitpref'];
     if ($resubmitpref) {
          //user has already requested for a resubmission
          echo '
          <p>You have made a request for a resubmission. You will shortly be
notified of our approval.</p>';
     } else {
          //user has not already requested for a resubmission
          //form for resubmission request
          echo '
          <form name="prefform" method="post" action="enrol.php"
class="textarea-form">
               <p>Please state the reason why you would like to change your
Big Brekky preferences. You will shortly be notified of our approval.</p>
               <textarea class="form-control" name="resubmitreason"
maxlength="500"></textarea>
               <input class="button" type="submit" name="submitrequest"
value="REQUEST">
          </form>';

     }
}
?>
```

4.3 Upload Completed Permission Form

```
<?php
$pdfFileExists = file_exists('permissionforms/'.$_SESSION['username'].'.pdf');
$pngFileExists = file_exists('permissionforms/'.$_SESSION['username'].'.png');
$jpgFileExists = file_exists('permissionforms/'.$_SESSION['username'].'.jpg');

//is student has already submitted a permission form
if ($pdfFileExists || $pngFileExists || $jpgFileExists) {
     echo '<p>You have already submitted a permission form. If required, you
can resubmit:</p>';
}
echo '
<form action="permission.php" method="post" enctype="multipart/form-data">
     <input class="file" type="file" name="file" id="file">
     <input class="button" type="submit" value="UPLOAD" name="submitfile">
</form>';

if (isset($_POST['submitfile'])) {
     //get file type and set file directory
     $fileType = strtolower(pathinfo(basename($_FILES["file"]["name"]),
PATHINFO_EXTENSION));
     $file_dir = 'permissionforms/'.$_SESSION['username'].'.'.$fileType;

     if (move_uploaded_file($_FILES["file"]["tmp_name"], $file_dir)) {
          //if file has been successfully uploaded to server
          alert('info', 'YOU PERMISSION FORM HAS BEEN UPLOADED');
     } else {
          alert('danger', 'THERE WAS AN ERROR UPLOADING YOUR FILE');
     }
}
?>
```

4.4 Viewing Student Profile

```
<head>
     <script>
          function showProfile(name) {
               //if input text is empty
               if (name.length == 0) {
                    //clear suggestions & exit function
                    document.getElementById('namesearch').innerHTML = '';
                    return;
               } else {
                    //create XMLHttpRequest object
                    var xmlhttp = new XMLHttpRequest();
                    //function called whenever readyState of request changes
                    xmlhttp.onreadystatechange = function() {
                         //if server response is ready
                         if (this.readyState == 4 && this.status == 200) {
                              //readyState = 4 -> request finished and
response is ready
                              //status = 200 -> returns "OK" for request
status
                              document.getElementById('namesearch').innerHTML
= this.responseText;
                         }
                    }
                    //send request to .php on the server - parameter name is
added
                    xmlhttp.open('GET', 'namesuggestions.php?name=' + name,
true);
                    xmlhttp.send();
               }
          }
     </script>
</head>

<?php
if (isset($_POST['savenotes'])) {
     //saving student notes
     $notes = trim(stripslashes(htmlspecialchars($_POST['notes'])));
     alert('info', 'STUDENT NOTES HAVE BEEN SAVED');

     DB::update('students', array(
          'notes' => $notes
     ), 'username = %s', $_POST['notesusername']);

}
?>

<div class="page-content">
     <h2>Search Student</h2>
     <form>
          <input type="text" class="form-control searchbar" placeholder="FULL
NAME" onkeyup="showProfile(this.value)">
          <div id="namesearch"></div>
     </form>

     <?php
     if (isset($_GET['username'])) {

          //retrieving student profile data from users table
          $usersProfile = DB::queryFirstRow('select * from users where
username = %s', $_GET['username']);
          //checking if username is valid
          if (isset($usersProfile)) {
               //retrieving student profile data from students table
               $studentsProfile = DB::queryFirstRow('select * from students
where username = %s', $_GET['username']);

               if ($studentsProfile['permitted'] == 1) {
                    //if student has submitted a parental permission form
                    $formfilepath =
'../student/'.$studentsProfile['formfilepath'];
                    //generate link for downloading submitted permission form
                    $formlink = '<a href="'.$formfilepath.'"
target="_blank">Download</a>';
               } else {
                    //student is yet to submit permission form
                    $formlink = 'Not Submitted';
               }

               echo '
               <table class="studentprofile-table">
                    <tr>
```

```
                    <th>NAME</th>
                    <td>'.$usersProfile['fname'].'
'.$usersProfile['lname'].'</td>
               </tr>
               <tr>
                    <th>S-NUMBER</th>
                    <td>'.$usersProfile['username'].'</td>
               </tr>
               <tr>
                    <th>HOUSE & PC</th>
                    <td>'.$studentsProfile['house'].'
'.$studentsProfile['pc'].'</td>
               </tr>
               <tr>
                    <th>PERMISSION FORM</th>
                    <td>'.$formlink.'</td>
               </tr>
          </table>

          <form name="notesform" method="post" action="students.php"
class="textarea-form">
               <h3>NOTES</h3>
               <textarea class="form-control" name="notes"
maxlength="500">'
               //display notes if it exists, otherwise display empty
string
               .(isset($studentsProfile['notes']) ?
$studentsProfile['notes'] : '').
               '</textarea>
               <input type="hidden" name="notesusername"
value="'.$usersProfile['username'].'">
               <input class="button" type="submit" name="savenotes"
value="SAVE NOTES">
          </form>';
     }
}
?>
</div>
```

4.5 XMLHttpRequest Response for Name Suggestions (Live Search Bar)

```
<?php
// get the name parameter from URL (text input)
$name = $_REQUEST['name'];
$suggestedNames = '';

//if name input isn't empty
if ($name !== "") {
     //retrieve 5 student names containing $name string in alphabetical order
     require('../globalfuncs.php');
     connectDB();
     $possibleStudents = DB::query('select username, concat(fname, " ", lname)
as name from users where concat(fname, " ", lname) like "'.$name.'%" and
usertype = "s" order by name limit 5');
     //concatenate each possible student's name onto $suggestedNames as a link
     foreach ($possibleStudents as $student) {
          //pass a GET variable (username) that'll be needed for displaying
student's profile
          $suggestedNames .= '<a class="namesuggestions"
href="students.php?username='.$student['username'].'">'.$student['name'].'</a>
';
          if ($suggestedNames !== '') {
               //add line break to display suggested names in rows
               $suggestedNames .= '<br>';
          }
     }
}

//output 'No Suggestion' if no suggestion was found or output correct names
echo $suggestedNames == '' ? '<a class="namesuggestions" id="nosuggestion">No
Suggestion</a>' : $suggestedNames;
?>
```

4.6 Viewing Students Cohort Preference List

```
<script>
$(document).ready(function(){
     //called if week or day select inputs are changed
     $('#pref-week, #pref-day').change(function(){
          //create XMLHttpRequest object
          var xmlhttp = new XMLHttpRequest();
          //function called whenever readyState of request changes
          xmlhttp.onreadystatechange = function() {
               //if server response is ready
               if (this.readyState == 4 && this.status == 200) {
                    //readyState = 4 -> request finished and response is
ready
                    //status = 200 -> returns "OK" for request status
                    //list the students preferences for the selected week and
day
                    document.getElementById('listpref').innerHTML =
this.responseText;
               }
          }
          //send request to .php on the server - parameters week & day inputs
are added
          xmlhttp.open('GET', 'listpref.php?week=' + $('#pref-week').val() +
'&day=' + $('#pref-day').val(), true);
          xmlhttp.send();
     }).change();
});
</script>

<div class="page-content">
     <h2>Student Preferences</h2>
     <form>
     <table class="weekday-table"><tr>
     <th>WEEK: </th>
     <td>
          <select class="form-control" name="pref-week" id="pref-week">
               <option value="a" selected>A</option>
               <option value="b">B</option>
          </select>
     </td>
     <th id="groupby-heading">DAY: </th>
     <td>
          <select class="form-control" name="pref-day" id="pref-day">
               <option value="mon" selected>MON</option>
               <option value="tue">TUE</option>
               <option value="wed">WED</option>
               <option value="thu">THU</option>
               <option value="fri">FRI</option>
          </select>
     </td>
     </tr></table>
     </form>
     <div id="listpref"></div>
</div>
```

4.7 XMLHttpRequest Response for Student Cohort Preference List Table

```
<?php
// get the week & day parameters from URL (select input)
$week = $_REQUEST['week'];
$day = $_REQUEST['day'];
$listpref = '';

//connect to database
require('../globalfuncs.php');
connectDB();

//display preferences in table
$listpref .= '<div class="table-responsive"><table class="table table-hover
preflist-table">';

//iterate through preference numbers 1 to 5
for ($prefnum=1; $prefnum<=5 ; $prefnum++) {
     //create new row in table for each preference number
     $listpref .= '<tr><th>Preference '.$prefnum.'</th>';
     //query database for every student prefence with the correct prefnum,
week, day
     $pref = DB::query('select concat(upper(u.lname), " ", u.fname) as name
from users u left join preferences p on p.username = u.username where
p.prefnum = %s and p.week = %s and p.day = %s order by u.lname', $prefnum,
$week, $day);

     //if at least one student preference exists for that prefnum, week, day
     if ($pref) {
          //turn 2d array into 1d arry and separate names by comma
          $prefSeparatedByComma = implode(',  ', array_column($pref,
'name'));
          $listpref .= '<td>'.$prefSeparatedByComma.'</td>';
     } else {
          $listpref .= '<td>NONE</td>';
     }
     $listpref .= '</tr>';
}
$listpref .= '</table></div>';

//output student preferences table
echo $listpref;
?>
```

4.8 Viewing Roster & Events Summary

```
<div class="banner general-banner">
     <h1>ROSTER</h1>
</div>

<div class="roster-tab">
     <a href="rosterpref.php">PREFERENCES</a>
     <a href="rosterroster.php" id="current-tab">ROSTER</a>
     <a href="rosteredit.php">EDIT</a>
</div>

<div class="page-content">
     <h2>Term Roster</h2>
     <div class="panel-group" id="accordion">
     <?php
     //finding the number of weeks in current term
     $currentDate = date('Y-m-d');
     $term = DB::query('select * from termweeks order by term');
     if ($currentDate < $term[1]['startdate']) {
          //current date is in term 1
          $numOfWeeks = $term[0]['weeknum'];
     } elseif ($currentDate < $term[2]['startdate']) {
          //current date is in term 2
          $numOfWeeks = $term[1]['weeknum'];
     } elseif ($currentDate < $term[3]['startdate']) {
          //current date is in term 3
          $numOfWeeks = $term[2]['weeknum'];
     } else {
          //current date is in term 4
          $numOfWeeks = $term[3]['weeknum'];
     }

     //array for caching student/mentor positions where enrolled > quota
     $exceedingPositions = array();

     //display roster in weekly blocks
     for ($weeknum=1; $weeknum<=$numOfWeeks; $weeknum++) {
          $weekLetter = $weeknum % 2 == 0 ? 'B' : 'A';
          //retrieve names of students and organisers allocated to events in
this week, grouped by rows of events - student and organiser names are
concatenated in their respective columns
          $events = DB::query('
                    select e.*, group_concat(distinct concat(upper(susers.lname), "
", susers.fname) order by susers.lname) as students, group_concat(distinct
concat(upper(ousers.lname), " ", ousers.fname) order by ousers.lname) as
organisers
                    from events e
                         left join studentevents s on e.eventid = s.eventid
                         left join users susers on s.username = susers.username
                         left join organiserevents o on e.eventid = o.eventid
                         left join users ousers on o.username = ousers.username
                    where e.week = '.$weeknum.'
                    group by e.eventid');

          if ($events) {
               //if there are events allocated during this week
               echo '
               <div class="panel panel-default">
               <div class="panel-heading">
                    <h3 class="panel-title">
                    <a data-toggle="collapse" data-parent="#accordion"
href="#collapse'.$weeknum.'">Week '.$weeknum.$weekLetter.'</a>
                    </h3>
               </div>
               <div id="collapse'.$weeknum.'" class="panel-collapse collapse">
                    <div class="panel-body table-responsive">
                    <table class="table table-hover roster-table" style="min-
width:1000px;">

                    <thead>
                         <tr>
                              <th rowspan="2"></th>
                              <th rowspan="2">TIME</th>
                              <th rowspan="2">PLACE</th>
                              <th rowspan="1" colspan="2">STUDENTS</th>
                              <th rowspan="1" colspan="2">MENTORS</th>
                         </tr>
                         <tr>
                              <th>ENROLLED</th>
                              <th>AVAILABLE</th>
                              <th>ENROLLED</th>
                              <th>AVAILABLE</th>
                         </tr>
                    </thead>
                    <tbody>';

               //list all allocated events in this week as table rows
               foreach ($events as $event) {
                    $starttime = date('H:i', strtotime($event['starttime']));
                    $endtime = date('H:i', strtotime($event['endtime']));
                    $studentList = $event['students'];
                    $organiserList = $event['organisers'];

                    $students = str_replace(',', '<br>', $studentList);
                    $organisers = str_replace(',', '<br>', $organiserList);
                    $numStudents = $studentList == '' ? 0 :
count(explode(',', $studentList));
                    $numOrganisers = $organiserList == '' ? 0 :
count(explode(',', $organiserList));
                    $studentPosAvailable = $event['studentquota'] -
$numStudents;
                    $organiserPosAvailable = $event['organiserquota'] -
$numOrganisers;

                    echo '
                    <tr>
                         <th>'.strtoupper($event['day']).'</th>
                         <td>'.$starttime.' - '.$endtime.'</td>
                         <td>'.$event['place'].'</td>
                         <td>'.$students.'</td>
                         <td>'.(string)$studentPosAvailable.'</td>
                         <td>'.$organisers.'</td>
                         <td>'.(string)$organiserPosAvailable.'</td>
                    </tr>';
                    //if available position less than 0, append into
$exceedingPositions array
                    if ($studentPosAvailable < 0) {
                         array_push($exceedingPositions, 'Week
'.strtoupper($event['week']).' '.strtoupper($event['day']).' Students');
                    }
                    if ($organiserPosAvailable < 0) {
                         array_push($exceedingPositions, 'Week
'.strtoupper($event['week']).' '.strtoupper($event['day']).' Organisers');
                    }
               }
               echo '
                    </tbody>
                    </table>
                    </div>
               </div>
               </div>';
          }
     }

     if (count($exceedingPositions) > 0) {
          echo '<p id="exceedingpositions"><b>EXCEEDING QUOTA:</b> '.join(",
",$exceedingPositions).'</p>';
     }
     ?>
</div>

</body>
</html>
```

4.9 Edit Roster (Event Details and Student & Organiser Allocations)

```
<?php
if (isset($_POST['submitrosteredit'])) {
     //if organiser has submitted the edit roster form
     if ($_POST['newevent'] == 1) {
          //organiser has added a new event
          //add event to events table
          DB::insert('events', array(
               'eventdate' => $_POST['eventdate'],
               'week' => $_POST['week'],
               'day' => $_POST['day'],
               'starttime' => $_POST['starttime'],
               'endtime' => $_POST['endtime'],
               'place' => $_POST['place'],
               'studentquota' => $_POST['studentquota'],
               'organiserquota' => $_POST['organiserquota']
          ));

          //newly inserted events's eventid
          $eventid = DB::queryFirstRow('select
last_insert_id()')['last_insert_id()'];
     } else {
          //organiser has edited an existing event
          $eventid = $_POST['eventid'];
          //update event in events table
          DB::update('events', array(
               'eventdate' => $_POST['eventdate'],
               'week' => $_POST['week'],
               'day' => $_POST['day'],
               'starttime' => $_POST['starttime'],
               'endtime' => $_POST['endtime'],
               'place' => $_POST['place'],
               'studentquota' => $_POST['studentquota'],
               'organiserquota' => $_POST['organiserquota']
          ), 'eventid = %s', $eventid);
     }
     //delete all students previously allocated to this event
     DB::delete('studentevents', 'eventid = %s', $eventid);
     //get the newly allocated students' names
     $studentnames = [$_POST['student1'], $_POST['student2'],
$_POST['student3'], $_POST['student4'], $_POST['student5'],
$_POST['student6'], $_POST['student7'], $_POST['student8']];
     foreach ($studentnames as $name) {
          //get student's username from his name
          $username = DB::queryFirstRow('select username from users where
lower(concat(fname, " ", lname)) = %s and usertype = "s"', $name)['username'];
          if ($username) {
               //if the entered name is valid
               //add newly allocated student to table
               DB::insert('studentevents', array(
                    'username' => $username,
                    'eventid' => $eventid
               ));
          }
     }

     //delete all organisers previously allocated to this event
     DB::delete('organiserevents', 'eventid = %s', $eventid);
     //get the newly allocated organisers' names
     $organisernames = [$_POST['organiser1'], $_POST['organiser2'],
$_POST['organiser3'], $_POST['organiser4']];
     foreach ($organisernames as $name) {
          //get student's username from his name
          $username = DB::queryFirstRow('select username from users where
lower(concat(fname, " ", lname)) = %s and usertype = "o"', $name)['username'];
          if ($username) {
               //if the entered name is valid
               //add newly allocated organiser to organisers
               DB::insert('organiserevents', array(
                    'username' => $username,
                    'eventid' => $eventid
               ));
          }
     }
     alert('info', 'ROSTER HAS BEEN SAVED');
}
?>

<div class="page-content">
     <h2>Select Week & Day</h2>
     <form name="weekdayform" method="post" action="rosteredit.php">
          <table class="weekday-table"><tr>
          <th>WEEK: </th>
          <td>
               <select class="form-control" name="week" id="week">
                    <?php
                    //finding the number of weeks in current term
                    $currentDate = date('Y-m-d');
                    $term = DB::query('select * from termweeks order by
term');
                    if ($currentDate < $term[1]['startdate']) {
                         //current date is in term 1
                         $numOfWeeks = $term[0]['weeknum'];
                    } elseif ($currentDate < $term[2]['startdate']) {
                         //current date is in term 2
                         $numOfWeeks = $term[1]['weeknum'];
                    } elseif ($currentDate < $term[3]['startdate']) {
                         //current date is in term 3
                         $numOfWeeks = $term[2]['weeknum'];
                    } else {
                         //current date is in term 4
                         $numOfWeeks = $term[3]['weeknum'];
                    }
                    //select options for week numbers in current term
                    for ($weeknum=1; $weeknum<=$numOfWeeks; $weeknum++) {
                         echo '<option
value="'.$weeknum.'">'.$weeknum.'</option>';
                    }
                    ?>
               </select>
          </td>
          <th id="groupby-heading">DAY: </th>
          <td>
               <select class="form-control" name="day" id="day">
                    <option value="mon" selected>MON</option>
                    <option value="tue">TUE</option>
                    <option value="wed">WED</option>
                    <option value="thu">THU</option>
                    <option value="fri">FRI</option>
               </select>
          </td>
          </tr></table>
          <input class="button" type="submit" name="submitweekday"
value="PROCEED">
     </form>

     <?php
     if (isset($_POST['submitweekday'])) {
          //if week and day submitted
          $week = $_POST['week'];
          $day = $_POST['day'];
          //retrieve event with the selected week and day
          $existingEvent = DB::queryFirstRow('select * from events where week
= %s and day = %s', $week, $day);
          if ($existingEvent) {
               //if event already exists on that day
               $newevent = 0;
               $eventid = $existingEvent['eventid'];
               $heading = 'Edit Existing';
               //create strings and arrays for automatic insertion of input
field values
               $eventdate = $existingEvent['eventdate'];
               $place = $existingEvent['place'];
               $starttime = $existingEvent['starttime'];
               $endtime = $existingEvent['endtime'];
               $studentquota = $existingEvent['studentquota'];
               $organiserquota = $existingEvent['organiserquota'];
```

```
                //initialising array for displaying the right number of student
name input fields
                $studentquotaselected = ['','','','','','','','','',''];
                $studentquotaselected[$studentquota-1] = 'selected';
                //initialising array for displaying the right number of
organiser name input fields
                $organiserquotaselected = ['','','','','','','','','',''];
                $organiserquotaselected[$organiserquota-1] = 'selected';
                //retrieve student names allocated to this event
                $studentnames = array_column(DB::query('select concat(fname, "
", lname) as name from users where username in (select username from
studentevents where eventid = %s) order by lname', $eventid), 'name');
                $numStudents = count($studentnames);
                //add empty strings to remaining positions to make an array of
8 elements
                $studentnames = array_merge($studentnames, array_fill(0, 8-
$numStudents, ''));
                //retrieve organiser names allocated to this event
                $organisernames = array_column(DB::query('select concat(fname,
" ", lname) as name from users where username in (select username from
organiserevents where eventid = %s) order by lname', $eventid), 'name');
                $numOrganisers = count($organisernames);
                //add empty strings to remaining positions to make an array of
8 elements
                $organisernames = array_merge($organisernames, array_fill(0, 4-
$numOrganisers, ''));
          } else {
                //if event does not already exist on that day
                $newevent = 1;
                $eventid = '';
                $heading = 'Add New';
                //create empty strings and arrays for input field values
                $eventdate = '';
                $place = $starttime = $endtime = '';
                $studentquotaselected = $organiserquotaselected =
['','','','','','','','','',''];
                $studentnames = ['','','','','','','','',''];
                $organisernames = ['','','','',''];
          }

          //generate the edit roster form
          echo'
          <h2>'.$heading.' Event</h2>
          <h3>Week '.$week.' '.strtoupper($day).'</h3>
          <form name="rosteredit" method="post" action="rosteredit.php"
class="form-horizontal roster-edit-form">
                <div class="form-group" style="width: 200px; margin: 0 auto
30px auto;">
                     <label class="control-label" for="eventdate">DATE</label>
                     <input type="date" class="form-control" name="eventdate"
id="eventdate" value="'.$eventdate.'" required>
                </div>
                <div class="form-group" style="width: 400px; margin: 0 auto
30px auto;">
                     <label class="control-label" for="place">PLACE</label>
                     <input type="text" class="form-control" name="place"
id="place" maxlength="80" value="'.$place.'" required>
                </div>
                <table class="roster-edit-table">
                     <tr>
                          <td><label class="control-label"
for="starttime">START TIME</label></td>
                          <td><label class="control-label" for="endtime">END
TIME</label></td>
                     </tr>
                     <tr>
                          <td><input class="form-control" type="time"
name="starttime" id="starttime" value="'.$starttime.'"></td>
                          <td><input class="form-control" type="time"
name="endtime" id="endtime" value="'.$endtime.'"></td>
                     </tr>
                </table>
                <table class="roster-edit-table" style="margin-bottom: 40px;">
                     <tr>
                          <td><label class="control-label"
for="studentquota">STUDENT QUOTA</label></td>
                          <td><label class="control-label"
for="organiserquota">ORGANISER QUOTA</label></td>
                     </tr>
                     <tr>
                          <td>
                          <select class="quota-select" name="studentquota"
id="studentquota">';
          //input select option for student quota
          for ($i=1; $i<=8; $i++) {
                echo '<option value="'.$i.'" '.$studentquotaselected[$i-
1].'>'.$i.'</option>';
          }
          echo '</select></td><td>
          <select class="quota-select" name="organiserquota"
id="organiserquota">';
          //input select option for organiser quota
          for ($i=1; $i<=4; $i++) {
                echo '<option value="'.$i.'" '.$organiserquotaselected[$i-
1].'>'.$i.'</option>';
          }
          echo'</select></td></tr><tr>
          <td><label class="control-label">STUDENT NAMES</label></td>
          <td><label class="control-label">ORGANISER NAMES</label></td>
          </tr><tr><td>';
          //input text fields for student names
          for ($i=1; $i<=8; $i++) {
                echo '<input type="text" class="form-control"
name="student'.$i.'" value="'.$studentnames[$i-1].'">';
          }
          echo '</td><td>';
          //input text fields for organiser names
          for ($i=1; $i<=4; $i++) {
                echo '<input type="text" class="form-control"
name="organiser'.$i.'" value="'.$organisernames[$i-1].'">';
          }
          echo '</td></tr></table>
          <input type="hidden" name="newevent" value="'.$newevent.'">
          <input type="hidden" name="week" value="'.$week.'">
          <input type="hidden" name="day" value="'.$day.'">
          <input type="hidden" name="eventid" value="'.$eventid.'">
          <input class="button" type="submit" name="submitrosteredit"
value="SUBMIT">
          </form>';
     }
     ?>
```

4.10 Displaying & Adding/Editing Reports

```
<?php
if (isset($_POST['submitreport'])) {
     $reportid = $_POST['reportid'];
     $eventid = $_POST['eventid'];
     $servings = $_POST['servings'];
     $feedback = $_POST['feedback'];
     $recordDate = date('Y-m-d H:i:s');

     //list of present students from saved report
     $presentStudents = isset($_POST['presentStudents']) ?
$_POST['presentStudents'] : [];
     //list of students that were meant to attend the event
     $eventStudents = array_column(DB::query('select username from
studentevents where eventid = %s', $eventid), 'username');
     //list of absent students from saved report
     $absentStudents = array_diff($eventStudents, $presentStudents);
     if ($reportid == 'null') {
          //new report was created
          //insert new report into reports table
          DB::insert('reports', array(
               'eventid' => $eventid,
               'servings' => $servings,
               'feedback' => $feedback,
               'recorddate' => $recordDate
          ));
          //newly inserted reports's rid
          $newreportid = DB::queryFirstRow('select
last_insert_id()')['last_insert_id()'];

          foreach ($absentStudents as $student) {
               //insert absent students into absences table
               DB::insert('absences', array(
                    'username' => $student,
                    'reportid' => $newreportid
               ));
          }

     } else {
          //existing report was edited
          //update report in reports table
          DB::update('reports', array(
               'servings' => $servings,
               'feedback' => $feedback
          ), 'reportid = %s', $reportid);

          //delete all absent students for this report from absences
          DB::delete('absences', 'reportid = %s', $reportid);
          foreach ($absentStudents as $student) {
               //insert absent students into absences table
               DB::insert('absences', array(
                    'username' => $student,
                    'reportid' => $reportid
               ));
          }
     }
     alert('info', 'REPORT HAS BEEN SAVED');
}

?>

<div class="page-content">
     <form name="addreportform" method="post" action="savereport.php">
          <?php
          //retrieving today's event
          $currentDate = date('Y-m-d');
          $eventToday = DB::queryFirstRow('select * from events where
eventdate = %s', $currentDate);
          //if event exists for today, allow organiser to create a new report
with corresponding POST variables
          if ($eventToday) {
               //retrieving today's report
               $reportToday = DB::queryFirstRow('select * from reports where
date(recorddate) = %s', $currentDate);
               if (!$reportToday) {
                    //if an organiser has not already submitted a report
today
                    echo '
                    <input type="hidden" name="eventid"
value="'.$eventToday['eventid'].'">
                    <input type="hidden" name="week"
value="'.$eventToday['week'].'">
                    <input type="hidden" name="day"
value="'.strtoupper($eventToday['day']).'">
                    <input class="button" type="submit" name="addreport"
value="ADD NEW REPORT">';
               } else {
                    echo '<p>A REPORT HAS ALREADY BEEN SUBMITTED TODAY</p>';
               }
          } else {
               echo '<p>THERE IS NO EVENT TODAY</p>';
          }
          ?>
     </form>

     <h2>Recent Reports</h2>
     <?php
     //retrieve 10 most recent existing reports completed in the current year
     $reports = DB::query('select * from reports where year(recorddate) = %i
order by recorddate desc limit 10', date('Y'));
     //if reports exist
     if ($reports) {
          echo '
          <div class="table-responsive">
               <table class="table table-hover reportsummary">
                    <thead>
                         <tr>
                              <th>WEEK</th>
                              <th>DAY</th>
                              <th>COMPLETED</th>
                              <th></th>
                         </tr>
                    </thead>
                    <tbody>';
          //iterating through each report to display data in table
          foreach ($reports as $report) {
               $event = DB::queryFirstRow('select * from events where eventid
= %s', $report['eventid']);
               $date = date_format(date_create($report['recorddate']), 'j M -
g:i A');
               echo '
                         <tr>
                              <td>'.$event['week'].'</td>
                              <td>'.strtoupper($event['day']).'</td>
                              <td>'.$date.'</td>
                              <td>
                                   <form name="editreportform" method="post"
action="savereport.php">
                                   <input type="hidden" name="reportid"
value="'.$report['reportid'].'">
                                   <input type="hidden" name="eventid"
value="'.$event['eventid'].'">
                                   <input type="hidden" name="week"
value="'.$event['week'].'">
                                   <input type="hidden" name="day"
value="'.strtoupper($event['day']).'">
                                   <input class="button" type="submit"
name="editreport" value="EDIT">
                                   </form>
                              </td>
                         </tr>';
          }
          echo '
                    </tbody>
               </table>
          </div>';
     } else {
          echo '<h3>NONE</h3>';
     }

     ?>
```

4.11 Form for Adding/Editing Reports

```
<div class="page-content savereport-content">
     <a href="report.php" class="button">BACK</a>
     <?php
     if (isset($_POST['addreport'])) {
          //if new report has been added
          $reportid = 'null';
          $eventid = $_POST['eventid'];
          $week = $_POST['week'];
          $day = $_POST['day'];

          //students attending this event
          $eventStudents = DB::query('select username, concat(fname, " ",
lname) as name from users where username in (select username from
studentevents where eventid = %s) order by lname', $eventid);
          //all students are initially set as absent
          $absentStudents = array_column($eventStudents, 'username');
     } else if (isset($_POST['editreport'])) {
          //if existing report is being edited
          $reportid = $_POST['reportid'];
          $eventid = $_POST['eventid'];
          $week = $_POST['week'];
          $day = $_POST['day'];

          //students attending this event
          $eventStudents = DB::query('select username, concat(fname, " ",
lname) as name from users where username in (select username from
studentevents where eventid = %s) order by lname', $eventid);
          //students previously marked as absent
          $absentStudents = array_column(DB::query('select username from
absences where reportid = %s', $reportid), 'username');
     } else {
          header('Location: report.php');
          exit();
     }
     ?>
     <form name="savereportform" method="post" action="report.php">
          <h2 style="margin-bottom: 60px;"><?php echo 'WEEK '.$week.'
'.$day; ?> REPORT</h2>

          <h3>1. SELECT PRESENT STUDENTS:</h3>
          <div class="rollcheckbox">
               <ul>
               <?php
               foreach ($eventStudents as $student) {
                    if (in_array($student['username'], $absentStudents)) {
                         //if student is/was absent
                         echo '
                         <li><label><input type="checkbox"
name="presentStudents[]" value="'.$student['username'].'">
 '.$student['name'].'</label></li>';
                    } else {
                         echo '
                         <li><label><input type="checkbox"
name="presentStudents[]" value="'.$student['username'].'" checked>
'.$student['name'].'</label></li>';
                    }
               }
               ?>
               </ul>
          </div>
          <?php
          $report = DB::queryFirstRow('select * from reports where reportid
= %s', $reportid);
          ?>
          <h3>2. NUMBER OF SERVINGS</h3>
          <input type="number" name="servings" step="1" class="form-control
servings" value="<?php echo $report['servings']; ?>">
          <h3>3. FEEDBACK</h3>
          <div class="textarea-form">
          <textarea class="form-control feedback" name="feedback"
maxlength="500"><?php echo $report['feedback']; ?></textarea></div>
          <input type="hidden" name="reportid" value="<?php echo
$reportid; ?>">
          <input type="hidden" name="eventid" value="<?php echo $eventid; ?>">
          <input class="button" type="submit" name="submitreport" value="SAVE
REPORT">
     </form>
</div>
```

4.12 Admin Query Engine

```
<?php
if (isset($_POST['submitquery'])) {
     //since we're catching errors, error handler not needed
     DB::$error_handler = false;
     DB::$throw_exception_on_error = true;
     try {
          $query = trim($_POST['query']);
          $queryresults = DB::query($query);
          echo '
               <div class="panel panel-success">
                    <div class="panel-heading">QUERY</div>
                    <div class="panel-body" id="query-panel">'.$query.'</div>
               </div>';

          //if the query returns a results table
          if (is_array($queryresults)) {
               $keys = array_keys($queryresults[0]);

               echo '
               <div class="panel panel-info">
                    <div class="panel-heading">RESULTS TABLE</div>
                    <div class="panel-body" style="padding-top:30px;">
                         <div class="table-responsive">
                              <table class="table table-hover retreats-
table">';
               //getting table headings
               echo '<tr>';
               foreach ($keys as $key) {
                    echo '<th>'.$key.'</th>';
               }
               echo '</tr>';

               //iterating through each row and displaying data
               foreach ($queryresults as $rowdata) {
                    echo '<tr>';
                    foreach ($rowdata as $key => $value) {
                         echo '<td>'.$value.'</td>';
                    }
                    echo '</tr>';
               }
               echo '
                              </table>
                         </div>
                    </div>';
          } else {
               echo '
               <div class="panel panel-info">
                    <div class="panel-heading">RESULTS TABLE</div>
                    <div class="panel-body">QUERY SUCCESSFUL</div>
               </div>';
          }
     } catch (MeekroDBException $e) {
          //catch meekroDB error
          echo '
               <div class="panel panel-danger">
                    <div class="panel-heading">ERROR</div>
                    <div class="panel-body" id="query-panel">'.$e-
>getMessage().'</div>
               </div>';
     }
}
?>
```

4.13 Sending Mails

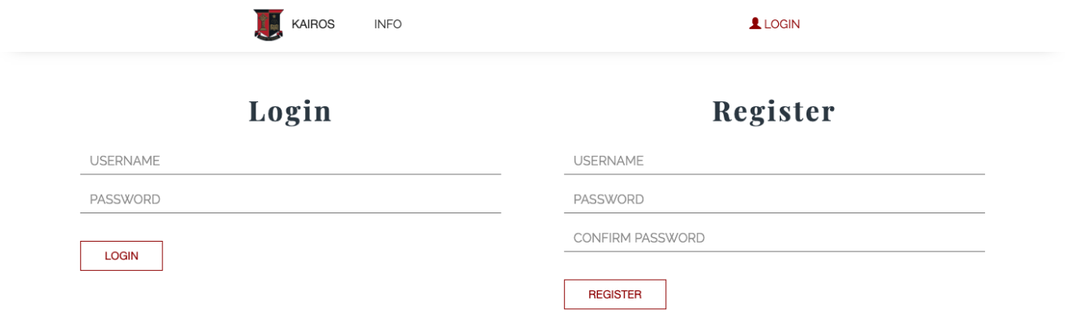
```
<?php
require_once('email/SMTP.php');
require_once('email/PHPMailer.php');
require_once('email/Exception.php');
use \PHPMailer\PHPMailer\PHPMailer;
use \PHPMailer\PHPMailer\Exception;

function sendVerificationEmail($address, $username, $token) {
     try {
          $mail = new PHPMailer(true); // Passing `true` enables exceptions
          //settings
          $mail->SMTPDebug = 0; // Enable verbose debug output
          $mail->isSMTP(); // Set mailer to use SMTP
          $mail->Host = 'smtp.gmail.com';
          $mail->SMTPAuth = true; // Enable SMTP authentication
          $mail->Username = 'Gtdigisol2020@gmail.com'; // SMTP username
          $mail->Password = 'securepassword'; // SMTP password
          $mail->SMTPSecure = 'ssl';
          $mail->Port = 465;
          $mail->setFrom('Gtdigisol2020@gmail.com', 'Digisol Test');

          //recipient
          //default 2FA email for testing purposes only
          $address = "s01575581@terrace.qld.edu.au";
          $mail->addAddress($address, $username); // Add a recipient
          $verificationlink =
'http://localhost/bigbrekky/index.php?username='.urlencode($username).'&token=
'.$token;

          //content
          $mail->isHTML(true); // Set email format to HTML
          $mail->Subject = 'Big Brekky Account Verification';
          $mail->Body = '
          <h1>BIG BREKKY ACCOUNT VERIFICATION</h1>
          <h3>Username: '.$username.'</h3>
          <p style="font-size: 16px;">Please click this link to verify your
account: </p>
          <a style="font-size: 16px; text-decoration: none;"
href="'.$verificationlink.'">VERIFY</a>';
          $mail->AltBody = "BIG BREKKY ACCOUNT VERIFICATION\r\n"."Please click
this link to verify your account:\r\n".$verificationlink;
          $mail->send();
     }
     catch(Exception $e) {
          echo 'Message could not be sent.';
          echo 'Mailer Error: '.$mail->ErrorInfo;
     }
}
?>
```

5.0 EVALUATION

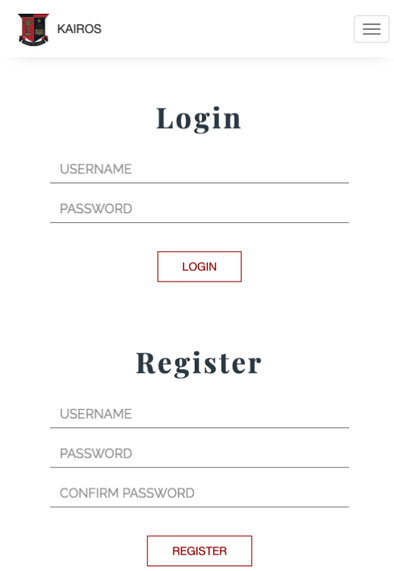
5.1 USER EXPERIENCE & USEABILITY

EXAMPLE UI DESIGN (LOGIN/REGISTRATION):

Laptop or Table View

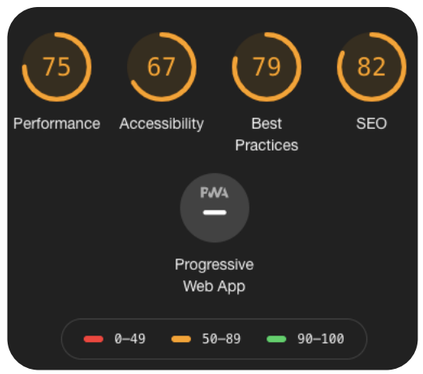


Mobile Phone View



Lighthouse Audit Report: Desktop

Lighthouse Audit Report: Mobile



EFFECTIVENESS

The website embraces a modern and minimalistic UI design with responsive layout to support mobile platforms, overall constituting a pleasing user experience. In particular, the navigation bar collapses into a hamburger menu on mobile platforms, while form inputs resize and reposition themselves according to the screen width, which was done by using Bootstrap and its extensive grid system. For instance, the login form stacks on top of the registration form when on a mobile device instead of having both sections laid out horizontally, further enhancing user experience through more spatial distribution of elements. Using the convenient navigation bar, users can easily navigate around the website with minimal number of clicks being required. The frequent pop-up alerts at the bottom of the screen (also implemented using bootstrap), alongside clear instructions for each form components provide documentation about the user requirements, while the Manual page aims to clarify any questions regarding the Big Brekky program at Terrace. Furthermore in the Student Profiles page, the live search bar assists organisers who have difficulty memorising students' names by suggesting similar possible names. Moreover, the implementation of AJAX XMLHttpRequest wireframe allows organisers to swiftly retrieve student data without having to reload the page each time, enhancing both efficiency and user experience. Overall, all three types of users are able to perform all of their desired actions without encountering any bugs, including event enrolment for students, editing the term roster and submitting daily reports for mentors, as well as directly querying the database for the admin. However, the lack images and icons, especially within the forms and the Manual page, limits the amount of visual aid for users, which could potentially hinder their understanding of certain components included in the digital solution. Nonetheless, through user-friendly features such as secure user-authentication and non-disruptive alert messages, this prototype provides a reliable system for Big Brekky management.

ACCESSIBLITY

This digital system offers some support for the disabled, especially those with visual impairments. Alongside the large Playfair and Railway fonts, the consistent black text and red buttons on a white background (which mirrors the colour scheme of the Terrace brand and complies with the College Style Guidelines Book) create high contrast for improved visibility. In the Roster pages for organisers, legibility is substantially improved by using a collapsible dropdown menu to prevent over-clustering of information and allow larger fonts. Coherent language and instructional diagrams have been incorporated to help dyslexic users avoid any confusion that may hinder their experience. Moreover, large clickable areas require less precision for activation, while the fixed navigation bar and auto-fill functionality in forms (e.g. saved student annotations and event details are auto-filled within the text field) minimise intensive user interaction such as scrolling and typing that can't be operated by someone with motor disabilities such as Parkinson's disease. As well, flashing effects have been avoided to prevent any sudden seizures for epilepsy patients. Lastly, in order to support those with language barriers, the 'lang' attribute in HTML was set to 'en' for English to allow accurate translations. According to the Lighthouse audit on Google Chrome, further improvements to accessibility could be made by specifying the 'alt' attribute of images in HTML, which could potentially be used for allowing screen-reading tools to describe images with text to blind users. However, this is not a major necessity as there is a limited number of images featured in the website.

SAFETY

Users often choose system functions by mistake and require options to leave the unwanted state. The Terrace logo in the navigation bar serves as a button redirecting users to the home page in case they get lost. Since errors are inevitable, organisers are able to edit every attribute of a previously submitted daily report, including the number of servings and list of students present. As well, organisers will only be allowed to add new reports during a Big Brekky session and the Week & Day headings for each report is pre-determined by a back-end code that compares the current date to the starting dates of the four school terms. If the user logins or registers mid-way through reading the Manual page, they will be redirected to that same page once authenticated. Moreover, if the user logs out but attempts to go back and revisit a page that's only accessible to a logged-in user, they'll be redirected to the homepage to prevent unauthorised access to sensitive data and content. Such error prevention measures will ameliorate both user experience and security measures implemented in this digital solution.

5.2 CRITERIA & IMPACTS

CRITERIA

PRESCRIBED CRITERIA

Student actions:

- ☒ [PC1] Select weekday preferences for Big Brekky enrolment and view selected preferences
- ☒ [PC2] Needs the ability to edit preferences
- ☒ [PC3] Download electronic copy of the parental permission form

Organiser actions:

- ☒ [PC4] View event summaries – include date, type and location, quota and current registration levels of participants
- ☐ [PC5] Apply participation caps or encourage more to get involved
  Although organisers can apply participation caps, they are not able to encourage specific students to get involved.
- ☒ [PC6] View and manage students' parental permission form
- ☐ [PC7] Print out an attendance roll and complete it online later when a digital device is accessible
  Despite there being a fully functioning roll in the 'Reports' page, there is no option for the organiser to print it out in hard copy. Also, the rolls display all students allocated to a certain event regardless of whether they are parent-authorised. This could potentially cause non-parent-authorised students to be marked as 'absent' despite not actually attending the event.
- ☒ [PC8] Annotate student profiles with performance notes

Admin actions:

- ☐ [PC9] Upload user data and promote users to organisers
  Admin is only able to upload user data via the SQL query engine. There was no need to implement the 'promote users to organisers' feature because in this prototype, organisers can register as organisers from the get-go.
- ☒ [PC10] Generate semester participation reports for inclusion in semester reporting

Data requirements:

- ☐ [PC11] Admin can add cohort studentbase data by uploading an appropriate .csv file. This dataset consists: s-number, first & last names, email address, house & PC group
  There is no user interface available for the admin to upload .csv files .
- ☒ [PC12] Rostered sessions have a minimum of: date, street address, start and end times, attendance cap, minimum required no. of organisers

User experience:

- ☐ [PC13] Comply with Australian Accessibility Standards and the College Style Guidelines booklet
  Refer to the checklist on the right.
- ☐ [PC14] Comply with the Australian Privacy Act (1998) – ensure that only authorised personnel can view personal data and inform users of their data being stored
  Admin can freely view, edit and manage personal data of students and mentors. Although they are technically 'authorised', there is no disclaimer/terms & conditions link on the registration page for informing new users about personal data storage.
- ☐ [PC15] Include appropriate attribution to data and images used. Comply with copyright law
  Due to copyright laws, only images from the Terrace website were used in the Big Brekky digital system. Hence, there was no need for acknowledgement.
- ☒ [PC16] Sanitise user input to prevent SQL injection attacks and securely store student information and passwords. Implement 2-factor authentication for secure registration

SELF-DETERMINED CRITERIA

Student actions:

- ☒ [SC1] Upload a digital copy of their completed parental permission form securely
- ☐ [SC2] Report issues encountered during Big Brekky sessions (anonymously if need be)
  Although the webpage is present, the form has not be fully programmed to function as required. This will be a component that'll be added for Version 2.0

Organiser actions:

- ☒ [SC3] Edit every aspect of the roster, including event details and the students & organisers allocated to each event. Organisers should also have the ability to add new events.

Admin actions:

- ☐ [SC4] Building a UI for admin to interact with the database instead of having to visit phpMyAdmin
  The only intuitive UI component provided to the admin is for generating semester participation reports. This is insufficient and may cause inconvenience for the admin due to limited knowledge about SQL commands when using the query engine.
- ☒ [SC5] Edit any aspect of the database via an SQL query engine
- ☒ [SC6] Admin is notified via email when the student submits a request for preference resubmission

☐ [SC7] Automatically generate a recommended term roster based on student preferences to reduce the workload of the organisers' when managing the roster
Due to time constraints and lack of required algorithmic knowledge, this component was unfortunately unable to be implemented. Automatically generating a term roster based on student preferences could be included in Version 2.0, which would significantly heighten user experience as the organiser no longer has to repetitively add an endless list of events via a single form.

☒ [SC8] Implement AJAX XMLHttpRequests for live database queries and webpage content updates without having to reload the page
☒ [SC9] UI is compatible with mobile platforms (responsive) and appealing to the human eye
☒ [SC10] Generation of efficient programming components, data elements and user interface
☒ [SC11] Error-proofing mechanisms to prevent defects caused by users
☒ [SC12] Compliant with useability principles (effectiveness, accessibility, safety)
☒ [SC13] Consider personal, social and economic implications to identify risks of the Big Brekky digital system

[PC13] Australian Accessibility Standards Checklist

Page titles:

- ☒ Must appear in the browser tab for all pages
- ☒ Must be appropriate for the page
- ☒ Must be different for each page

Alt text:

- ☐ Must be used for all content images (except decorative images)
- ☐ Attribute is set to null for decorative images
- ☐ Appropriately describes the content of the image to which it relates

  As explained above in the 'Accessibility' section, no alt texts were used. Specifying the 'alt' attribute of images in HTML could potentially be used for allowing screen-reading tools to describe images with text to blind users, which would significantly enhance user accessibility.

Headings:

- ☒ Are on every page (at least one)
- ☒ Levels on each page have a meaningful hierarchy

Zooming of pages:

- ☒ Results in correct display of the page with no horizontal scrolling
- ☒ Allows all buttons to remain visible

Non-mouse navigation (keystrokes or tabs):

- ☒ Of page is in a logical order
- ☐ Allows access to all page elements

  There was minimal implementation of non-mouse navigation. Hence, not all the page elements could be accessed via this form of user interaction.

Error messages:

- ☒ Are clear and specific
- ☒ Do not cause the form to be completely reset.

IMPACTS

Personal Impacts

This digital Big Brekky system will provide students with a mechanism for conveniently submitting their weekday preferences online directly to the Big Brekky management team. They will also be able to instantly contact the admin if they desire to request for a resubmission. However, such ease comes with a price. Student's privacy and security of personal data is jeopardised by registering their accounts, as this website is still at its early stages of development; thus, minimal security measures have been implemented. Furthermore, the digital record of student profiles and rolls will serve as a secondary source of information for organisers, in case they lose their paper copies. As well, by being able to instantly analyse any student's annotations at any time via the web, organisers will be able to quickly derive methods for enhancing pupil's experiences.

Social Impacts

There are no major social impacts as this website will not be implemented for use by the wider general public, instead merely within Terrace.

Economic Impacts

If the school implements this digital solution, they will not have to waste money in hiring a separate web-development company to build and maintain the system. The digital solution is designed in a way such that only one administrator is required for managing the system, hence requiring minimal expenses for Terrace as the employee. By transferring a paper-spreadsheet system into an entirely digital online application, less money will be wasted on printing. As well, this digital system will allow highly efficient management of the Big Brekky program to spend more time on ameliorating student experiences – time is money!

Legal (Digital Copyright) Impacts

Permission to store students' personal data has been granted by their guardians (already being stored on databases like TASS). No images were used other than those retrieved from the official Terrace website. Therefore, digital copyright issues with the application are not anticipated.

REFERENCES

Coggle. (n.d.). *Coggle*. Retrieved June 5, 2019, from Coggle: https://coggle.it/

Gerber, R. (2018, March 5). *9 security tips to protect your website from hackers*. Retrieved June 14, 2019, from Creative Bloq: https://www.creativebloq.com/web-design/website-security-tips-protect-your-site-7122853

MeekroDB. (n.d.). *MeekroDB Documentation*. Retrieved June 2020, from MeekroDB: https://meekro.com/docs.php

PHPMailer. (n.d.). *PHPMailer*. Retrieved June 2020, from GitHub: https://github.com/PHPMailer/PHPMailer

QCAA. (2019). *Digital Solutions 2019 v1.2*. Retrieved June 14, 2019, from QCAA: http://www.wonko.info/ds/qcaa/snr_digital_solutions_19_ia2_asr_high.pdf

5.3 USER-TESTING & RECOMMENDATIONS

Component	Expected Outcome	User Feedback & Recommendations (Nikhil Madala)
General UI design and responsivity	User interface was taken into great consideration through the use of sophisticated CSS stylings for elegant design and Bootstrap classes for structured responsivity. By applying similar UI elements to the official Terrace website, this digital solution mirrored the modern and professional appearance through brand tie-in.	The website looks very clean and professional. Elements like the navigation bar and form inputs restructure themselves depending on the screen size, supporting mobile platforms. I especially liked your modern and simplistic approach to the UI (e.g. plain white background). Brand tie-in is obvious, featuring the consistent red-white colour scheme and fonts that are used in every other Terrace document/website.
Big Brekky manual page	User can read general information about the Big Brekky Program at Terrace, including a typical morning procedure and risk management. There will be a Bible quote by John in the middle of the webpage (retrieved from the Terrace website), above a separate section that displays contact details and location map.	Wow even better UI design! I especially love the bible quote section – pretty cool. Paragraphs are very informative, although I'm not too sure if this component is necessary – I don't imagine many students/mentors reading it. The contact details and location map is responsive for mobile platforms.
User authentication (registration, 2FA, login)	For registration, a user must type in his/her username (s-number for students and first & last names for organisers) and confirm the password. If the username is valid, then (s)he'll be sent an account verification email. (S)he must click the verification link before proceeding to login. In the system's current version, admin can login without having to register.	The 'Username' field should specify 's-number' for students; I thought I could make up my own username. Not being aware that the page was loading while it was sending the account activation email, I was confused as to why the screen 'froze' – please notify the users to wait. After registration/login, users should be redirected to the homepage instead of the previous page they were on for easier navigation.
Student: preference selection & parental permission form	Once students submit their weekday preferences, the webpage will display a summary of their selections and an option to request for a resubmission by providing a reason (admin will be notified via email). Subsequently, students will be allowed to download the parental permission form. Once the form is completed, it can be scanned and uploaded electronically for convenient storage of permission forms.	It was easy to choose the preferences; however, the UI design looked a bit generic and dull. Initially, I thought the resubmission request form was mandatory – it should clearly be labelled as optional. As well, students should get a confirmation email of their resubmission request to keep a digital record. Downloading the parental permission form is effortless and I especially like how only users who have submitted their preferences have access to it. Your website should prevent students who have requested for a resubmission from downloading the parental permission form as their parents could potentially authorise retreat selections that are subject to change. Definitely implement methods of checking whether the uploaded form is in fact, a completed form and not some random image file.
Organiser: student profile management (saving annotations and downloading students' permission form)	Organisers can search for student names using a live search bar that provides name suggestions. Once a student is selected, organisers can view his profile and edit student annotations. There is also an option to download the selected student's parental permission form if it has been submitted.	All functionality works swiftly. The search bar would be extremely convenient for organisers who don't know students' full names. I like how the number of name suggestions is limited to 5 so that the screen space doesn't get too confined. Student annotations save without a problem. Also, being able to download a student's permission form electronically instead of searching through piles of paper forms is highly convenient, resulting in a much more pleasing user experience. Nothing in particular that I dislike about this component ☺.
Organiser: roster summary and edit events	Mentors can view the term roster (starting dates, enrolled students and organisers, quotas, location, etc.), as well as the number of available student/organiser positions for each Brekky event during the current term.	Again, everything works perfectly fine except one minor issue. My only gripe is that in the 'Available Positions' column of the term roster tables, the numeric figures are negative whenever the enrolled number is greater than the quota. Theoretically, you can't have a negative number of positions available; hence they should be changed to 0.
Organiser: daily reports	Organisers can edit or add new reports. Either way, they'll have to provide the number of servings, then mark each student attending the Big Brekky session as absent or present. Feedback or comments are optional.	Although adding and editing reports work without any bugs, I have a few suggestions. Organisers shouldn't be allowed to edit the number of servings for accuracy purposes (memory isn't always reliable). I love how organisers can only add retreats during a Big Brekky session, but that locking-out functionality should also be implemented for editing reports.
Admin: database query engine	Admin can directly interact with and query the database as they normally would in phpMyAdmin. The results table and error messages are displayed in a neat format.	Assuming that the admin is fluent with SQL queries is risky as they could potentially damage the database. Instead, there should be a more intuitive user interface (e.g. a form) for executing simple commands like adding new mentors.

NOTE: some words from Nikhil's feedback and recommendations have been edited/omitted for clarity

FURTHER RECOMMENDATIONS

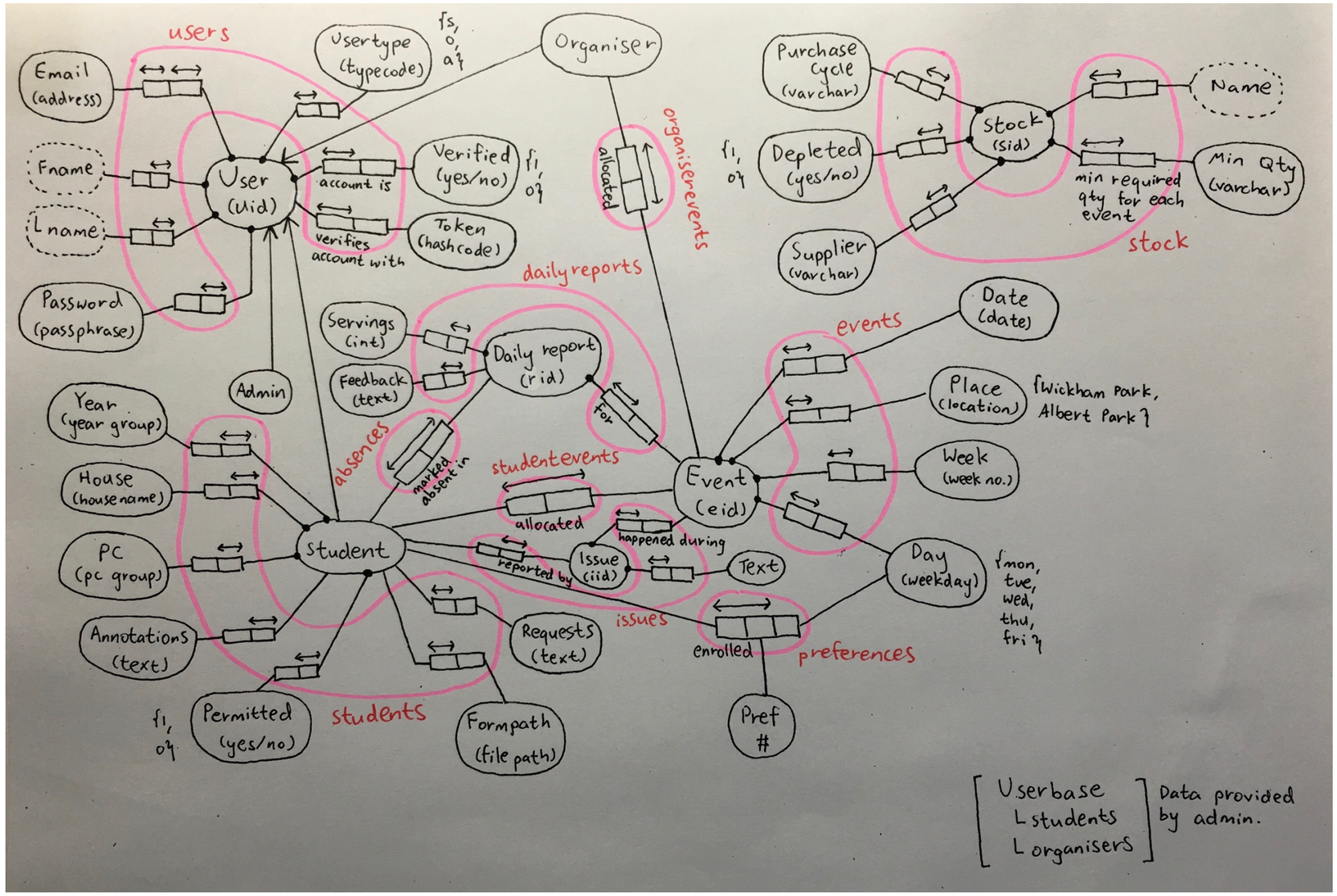
There are further significant improvements to be made. Firstly, each page has to download a set of google fonts that were used as an alternative to pre-installed fonts for a modern design. This hindered the efficiency of the website, especially when the Wi-Fi connection was moderately slow. This could be overcome by importing only the required font families and sizes, which will substantially increase efficiency. As well, the homepage features a massive banner image that is over 3000 pixels wide to support desktop screens, which can also slow down the website's speed. This can be resolved by utilising CSS to scale images depending on the viewport size of the screen. In general, more time must be spent on cleaning up the code to reduce redundancy (e.g. unnecessarily repeated code), which could not be done due to time constraints.

Despite several security measures hashing passwords, sanitising text inputs, and using POST method for sending forms, this website is vulnerable to more significant and deleterious cyber-attacks such as malware, phishing schemes and stolen data as I, the developer, should prevent by implementing mechanisms to impede network breaches. This will increase privacy and security of extremely sensitive information such as students' personal data, which could potentially be stolen.

As well, the future version of this website must use HTTPS protocol, which further prevents hackers from changing information on the page to accumulate personal information from users. Protection against XSS (cross-site scripting) attacks must also be considered for security against malicious injected JavaScript code that could alter page content or send vulnerable information to the attacker. Furthermore, if the Big Brekky system was to be extended to be fully operational, components such as the report-issue form for students should be programmed to be functional. Such improvements will allow an entirely digital system for Big Brekky management at Terrace.

6.0 APPENDICES

Appendix 1: Initial ORM digram



Appendix 1: Initial ORM digram