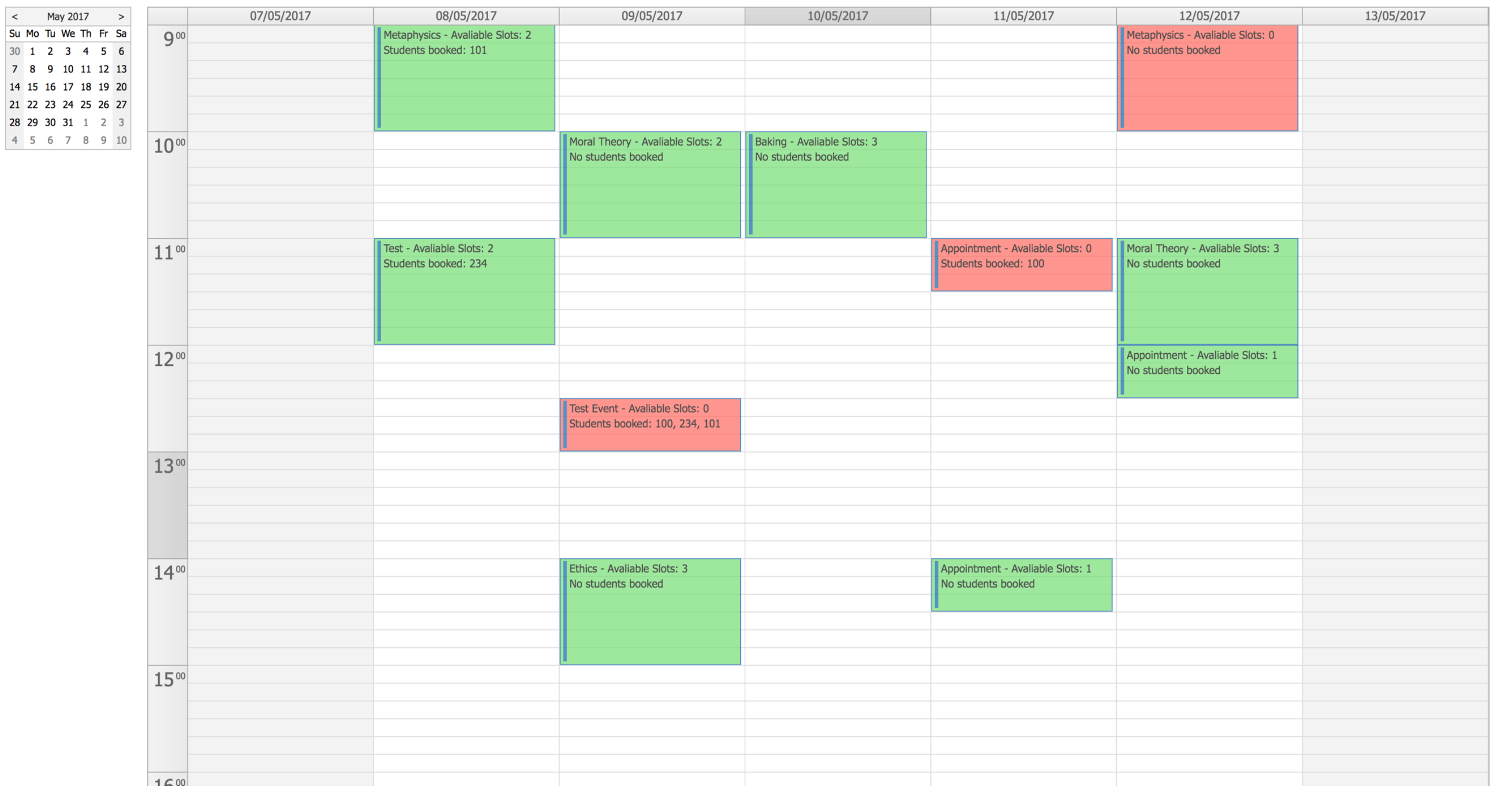
Huseyin Arpalikli - 13153439

CW1 Report

For this assignment, I have chosen to use the DayPilot Calendar API Library for my calendar. The reason I have decided to use this library is that I did not feel it would be efficient to 'reinvent the wheel' and create a calendar table from scratch. The API I have used is very flexible and has allowed me to change and add features to the event calendar. DayPilot Calendar is an AJAX event calendar widget, which makes use of jQuery and supports drag and drop operations (event creation, moving, resizing), context menus, event bubbles and custom event properties (menu, colour, HTML). The fact that custom event properties were possible has influenced my decision to



Metaphysics - Avaliable Slots: 2
Students booked: 101

Metaphysics - Avaliable Slots: 0
No students booked

Moral Theory - Avaliable Slots: 2
No students booked

Baking - Avaliable Slots: 3
No students booked

Test - Avaliable Slots: 2
Students booked: 234

Appointment - Avaliable Slots: 0
Students booked: 100

Moral Theory - Avaliable Slots: 3
No students booked

Appointment - Avaliable Slots: 1
No students booked

Test Event - Avaliable Slots: 0
Students booked: 100, 234, 101

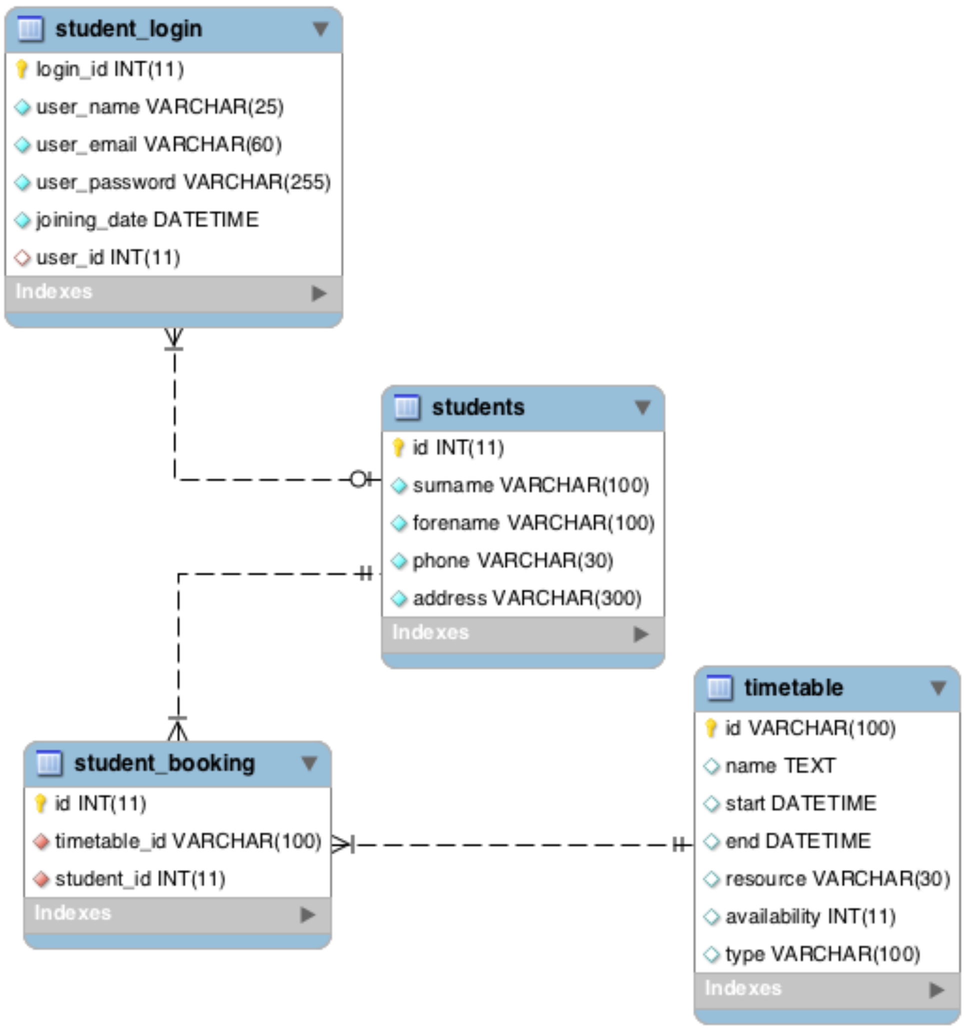
Ethics - Avaliable Slots: 3
No students booked

Appointment - Avaliable Slots: 1
No students booked

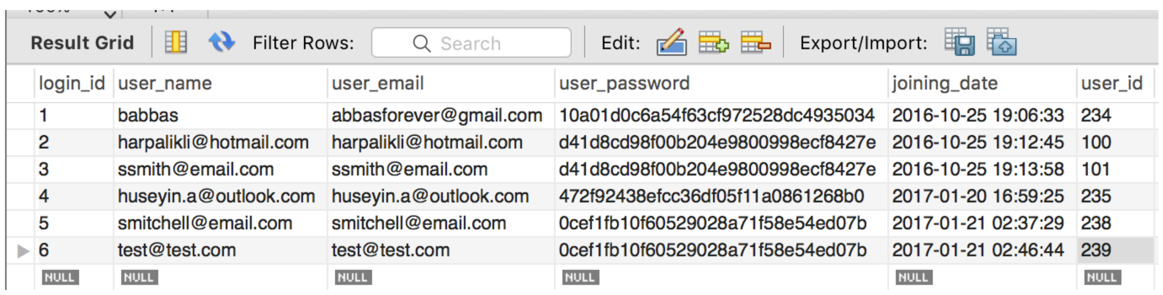
use the DayPilot API, as this meant that I could make changes in the calendar and custom additional features based on the assignment brief requirements. The events in the calendar are stored in a JSON format which means the data they can hold can be customised making the API flexible. The calendar has provided all the features I require and thus, I feel that the use of the library was a good decision to keep the Socrates appointment system efficient, dynamic and responsive.
The above screenshot shows my Socrates system in action with the use of the DayPilot API library for the calendar widget.

Huseyin Arpalikli - 13153439

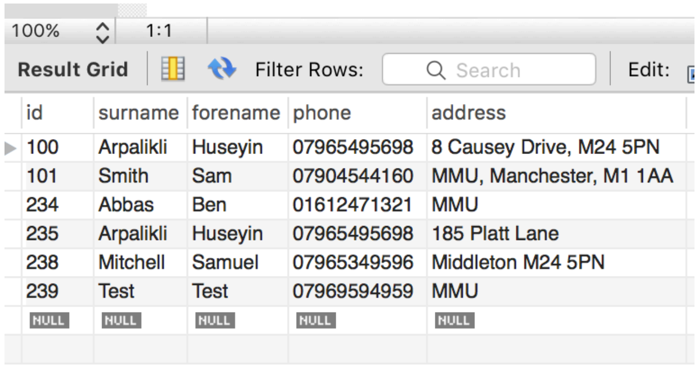
Regarding table structures chosen for the database, I have aimed for my data to be normalised to Third Normal Form (3NF). This was so that repeated data was avoided within the database and the database tables are as efficient as possible. I have made use of auto increments, primary and foreign keys. I have made sure that each table needed in the assignment does not repeat data between tables. As well as this, the use of 3NF has meant that without having repeated data, I could efficiently select data I needed by just linking foreign keys to primary keys between tables and thus, having tables that retrieved only the necessary data. On the left is an entity relationship diagram of my database system describing the links between the different tables within my database schema.



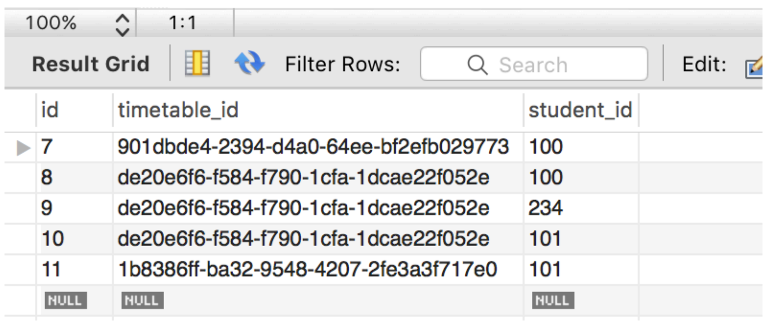
In the following screenshots, I have created and populated the necessary database tables. This complies with task 1 of section 0.3.



login_id	user_name	user_email	user_password	joining_date	user_id
1	babbas	abbasforever@gmail.com	10a01d0c6a54f63cf972528dc4935034	2016-10-25 19:06:33	234
2	harpalikli@hotmail.com	harpalikli@hotmail.com	d41d8cd98f00b204e9800998ecf8427e	2016-10-25 19:12:45	100
3	ssmith@email.com	ssmith@email.com	d41d8cd98f00b204e9800998ecf8427e	2016-10-25 19:13:58	101
4	huseyin.a@outlook.com	huseyin.a@outlook.com	472f92438efcc36df05f11a0861268b0	2017-01-20 16:59:25	235
5	smitchell@email.com	smitchell@email.com	0cef1fb10f60529028a71f58e54ed07b	2017-01-21 02:37:29	238
6	test@test.com	test@test.com	0cef1fb10f60529028a71f58e54ed07b	2017-01-21 02:46:44	239
NULL	NULL	NULL	NULL	NULL	NULL

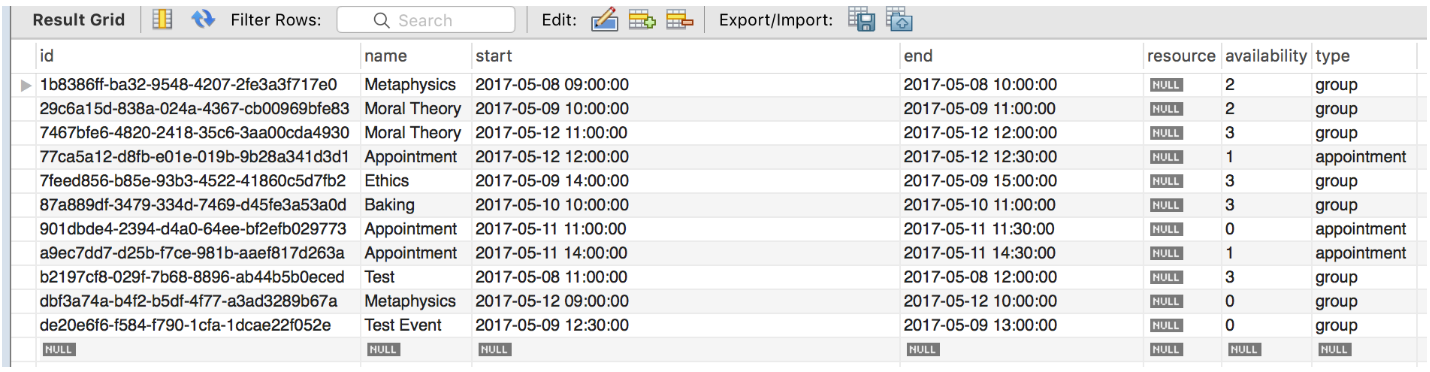


id	surname	forename	phone	address
100	Arpalikli	Huseyin	07965495698	8 Causey Drive, M24 5PN
101	Smith	Sam	07904544160	MMU, Manchester, M1 1AA
234	Abbas	Ben	01612471321	MMU
235	Arpalikli	Huseyin	07965495698	185 Platt Lane
238	Mitchell	Samuel	07965349596	Middleton M24 5PN
239	Test	Test	07969594959	MMU
NULL	NULL	NULL	NULL	NULL



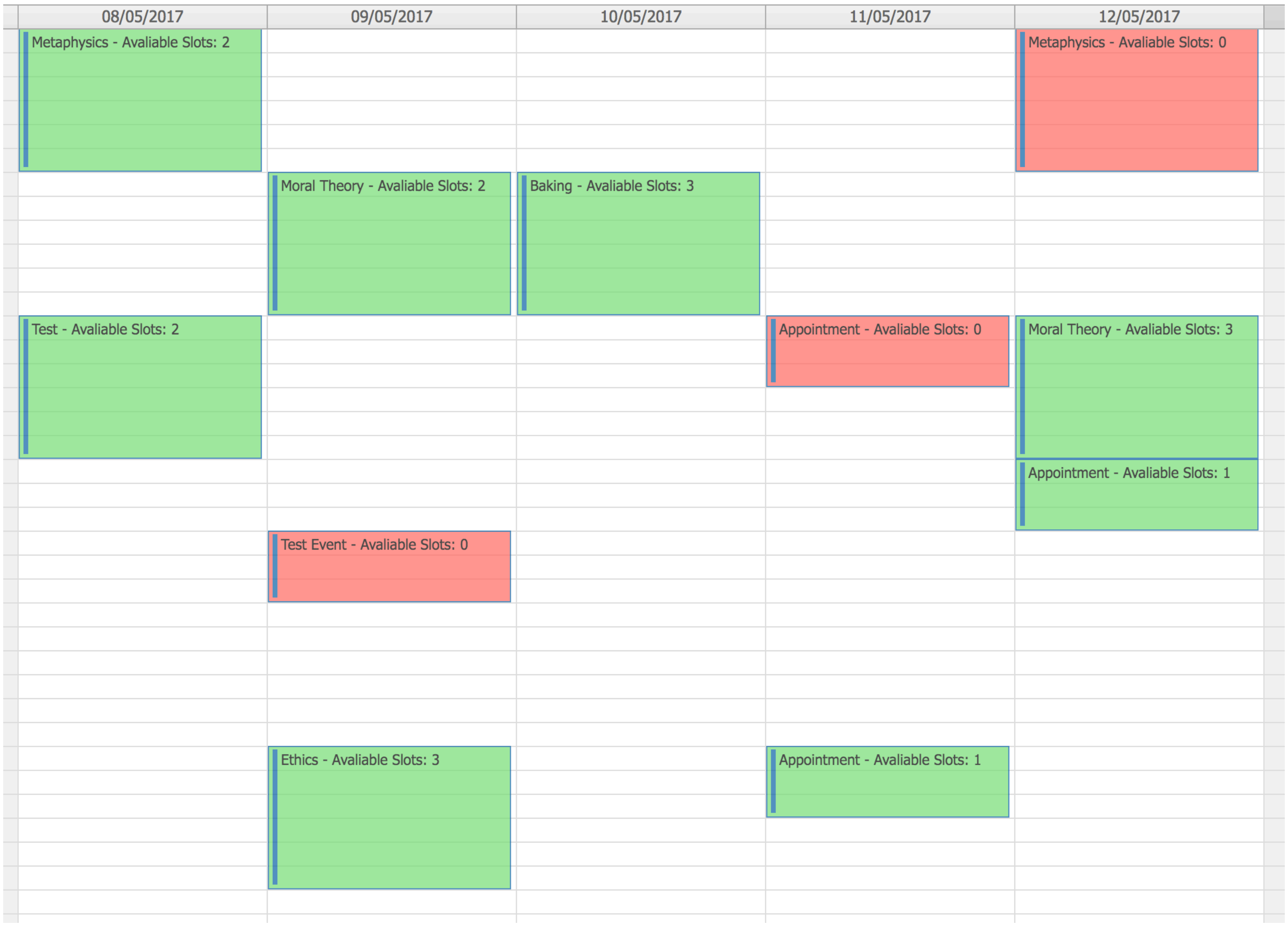
id	timetable_id	student_id
7	901dbde4-2394-d4a0-64ee-bf2efb029773	100
8	de20e6f6-f584-f790-1cfa-1dcae22f052e	100
9	de20e6f6-f584-f790-1cfa-1dcae22f052e	234
10	de20e6f6-f584-f790-1cfa-1dcae22f052e	101
11	1b8386ff-ba32-9548-4207-2fe3a3f717e0	101
NULL	NULL	NULL

Huseyin Arpalikli - 13153439

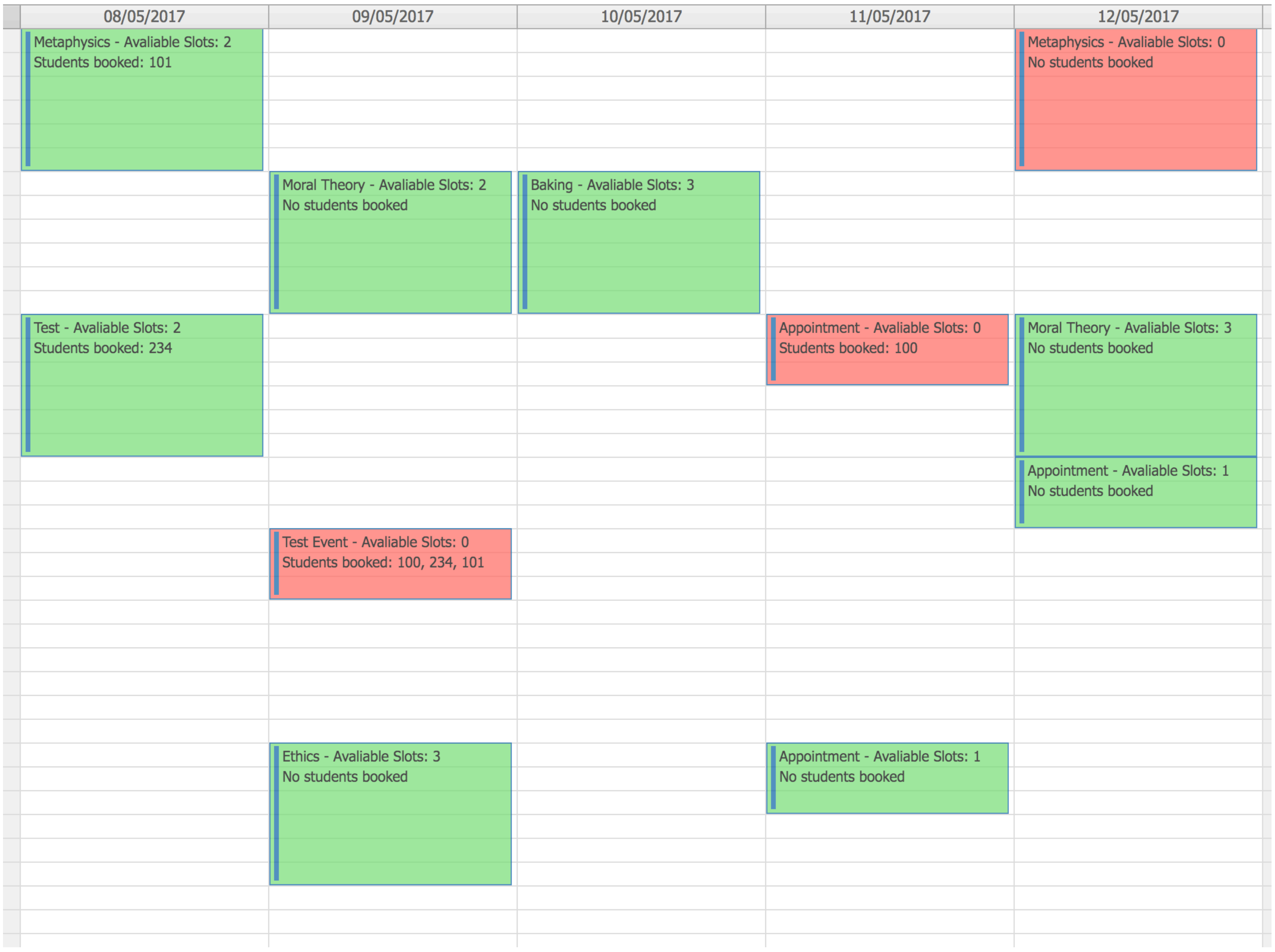


id	name	start	end	resource	availability	type
1b8386ff-ba32-9548-4207-2fe3a3f717e0	Metaphysics	2017-05-08 09:00:00	2017-05-08 10:00:00	NULL	2	group
29c6a15d-838a-024a-4367-cb00969bfe83	Moral Theory	2017-05-09 10:00:00	2017-05-09 11:00:00	NULL	2	group
7467bfe6-4820-2418-35c6-3aa00cda4930	Moral Theory	2017-05-12 11:00:00	2017-05-12 12:00:00	NULL	3	group
77ca5a12-d8fb-e01e-019b-9b28a341d3d1	Appointment	2017-05-12 12:00:00	2017-05-12 12:30:00	NULL	1	appointment
7feed856-b85e-93b3-4522-41860c5d7fb2	Ethics	2017-05-09 14:00:00	2017-05-09 15:00:00	NULL	3	group
87a889df-3479-334d-7469-d45fe3a53a0d	Baking	2017-05-10 10:00:00	2017-05-10 11:00:00	NULL	3	group
901dbde4-2394-d4a0-64ee-bf2efb029773	Appointment	2017-05-11 11:00:00	2017-05-11 11:30:00	NULL	0	appointment
a9ec7dd7-d25b-f7ce-981b-aaef817d263a	Appointment	2017-05-11 14:00:00	2017-05-11 14:30:00	NULL	1	appointment
b2197cf8-029f-7b68-8896-ab44b5b0eced	Test	2017-05-08 11:00:00	2017-05-08 12:00:00	NULL	3	group
dbf3a74a-b4f2-b5df-4f77-a3ad3289b67a	Metaphysics	2017-05-12 09:00:00	2017-05-12 10:00:00	NULL	0	group
de20e6f6-f584-f790-1cfa-1dcae22f052e	Test Event	2017-05-09 12:30:00	2017-05-09 13:00:00	NULL	0	group
NULL	NULL	NULL	NULL	NULL	NULL	NULL

As required by task 2 of section 0.3 in the assignment brief, my student view calendar displays data that is at least as complicated as Table 1 in the brief.

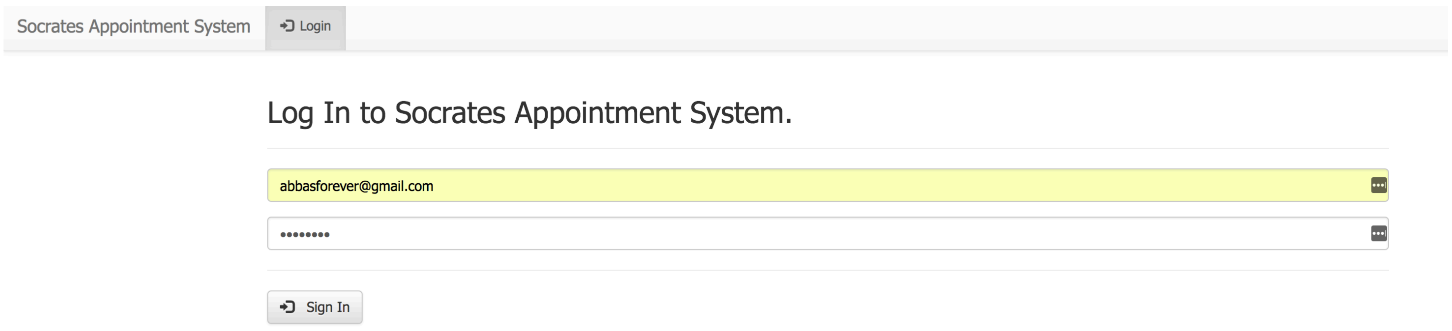

08/05/2017
09/05/2017
10/05/2017
11/05/2017
12/05/2017
Metaphysics - Avaliable Slots: 2
Metaphysics - Avaliable Slots: 0
Moral Theory - Avaliable Slots: 2
Baking - Avaliable Slots: 3
Test - Avaliable Slots: 2
Appointment - Avaliable Slots: 0
Moral Theory - Avaliable Slots: 3
Appointment - Avaliable Slots: 1
Test Event - Avaliable Slots: 0
Ethics - Avaliable Slots: 3
Appointment - Avaliable Slots: 1

The screenshot below shows that the data in my Socrates view calendar is at least as complex as Table 2 from the brief, as required by task 1 of section 0.4.



08/05/2017	09/05/2017	10/05/2017	11/05/2017	12/05/2017
Metaphysics - Avaliable Slots: 2 Students booked: 101				Metaphysics - Avaliable Slots: 0 No students booked
	Moral Theory - Avaliable Slots: 2 No students booked	Baking - Avaliable Slots: 3 No students booked		
Test - Avaliable Slots: 2 Students booked: 234			Appointment - Avaliable Slots: 0 Students booked: 100	Moral Theory - Avaliable Slots: 3 No students booked
				Appointment - Avaliable Slots: 1 No students booked
	Test Event - Avaliable Slots: 0 Students booked: 100, 234, 101			
	Ethics - Avaliable Slots: 3 No students booked		Appointment - Avaliable Slots: 1 No students booked	

For task 2 of section 0.4, I have decided to split this requirement down into two. To comply with the password protection requirement, I have created a login system that uses the email and password stored in the database and then uses PHP sessions to keep a user is logged in. Before a student has access to Socrates Student View, they will need to log in (logging in sets a session with the user id and email). To further protect the users from hacking, I have also hashed password into the MD5 format within the database. Below is the login screen displayed to a user that is not logged into the student view. AJAX is used to process the login PHP file and redirects the user to the Socrates student view.

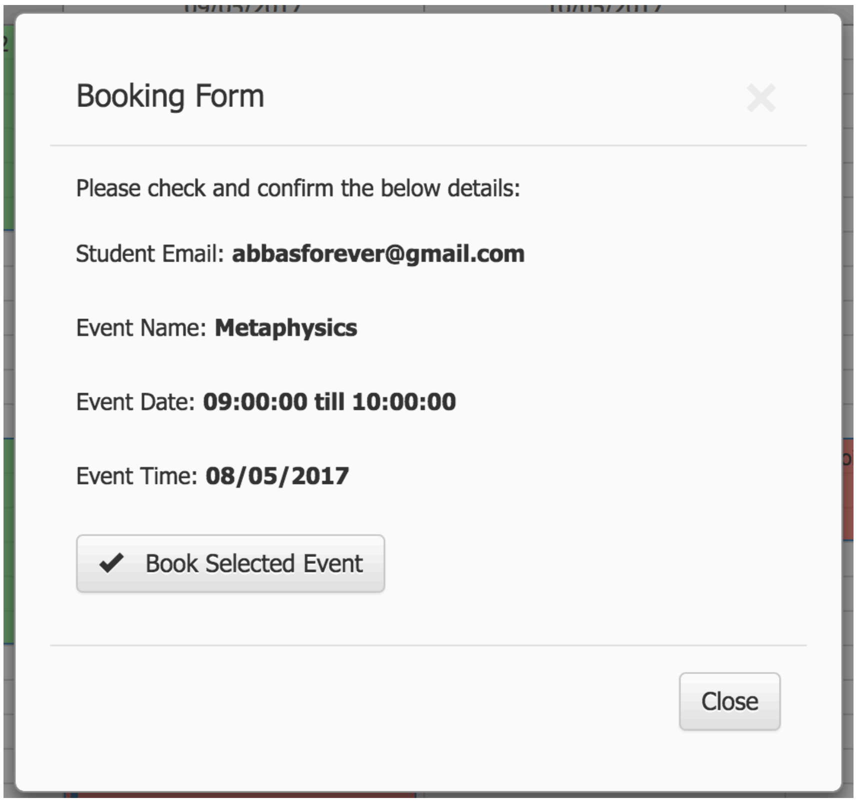


Log In to Socrates Appointment System.

abbasforever@gmail.com

••••••••

The above login screen uses the jQuery validator to ensure fields are filled in and to ensure the email is in the correct format.



Booking Form

Please check and confirm the below details:

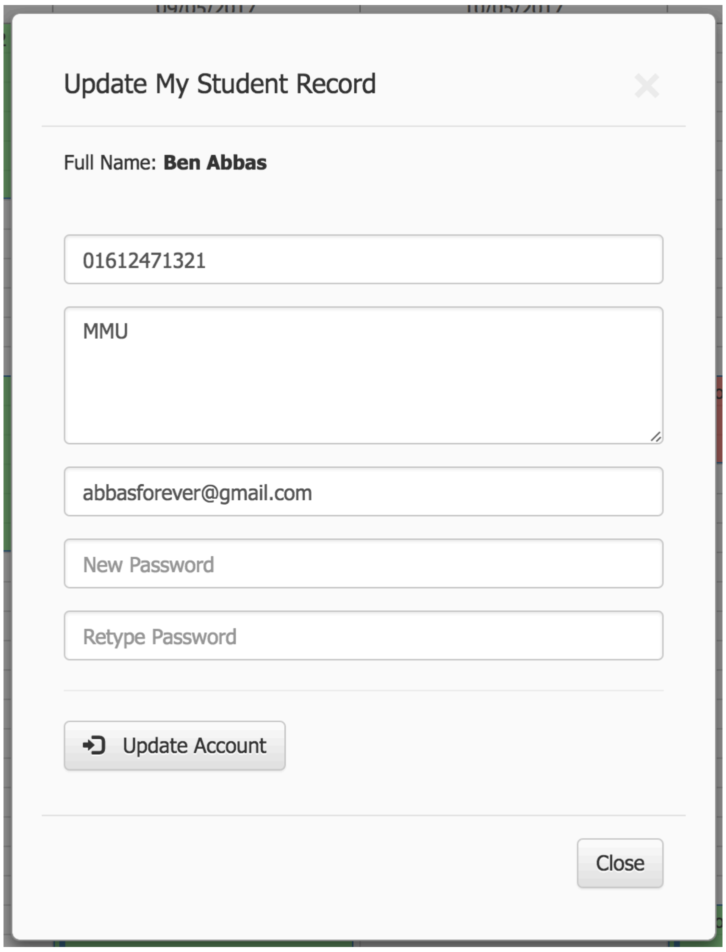
Student Email: **abbasforever@gmail.com**

Event Name: **Metaphysics**

Event Date: **09:00:00 till 10:00:00**

Event Time: **08/05/2017**

After a student is logged in a user can book into an event. When a user clicks on an event, a modal is displayed; AJAX is used to process PHP to get event details in a JSON format, along with session retrieval (to retrieve the logged in students email address and ID). Once the data is retrieved, the values are added to the form labels and fields on the modal and then the modal is displayed to the user. The screenshot on the left shows the modal with the user details present. When the user submits the booking, the record is added to the database table student_booking, which contains the student's ID, the event's ID as well as a generated booking ID. The availability is also updated to reflect the booking. In the example on the left, the user, Ben Abbas, has clicked on the Metaphysics event. Upon clicking the event, the event data is retrieve using arguments (event ID) in the DayPilot API, and this argument is then passed into the AJAX to retrieve the full event data in a JSON format. This is then passed into JavaScript variables which are used to populate the HTML.


Update My Student Record

Full Name: **Ben Abbas**

01612471321

MMU

abbasforever@gmail.com

New Password

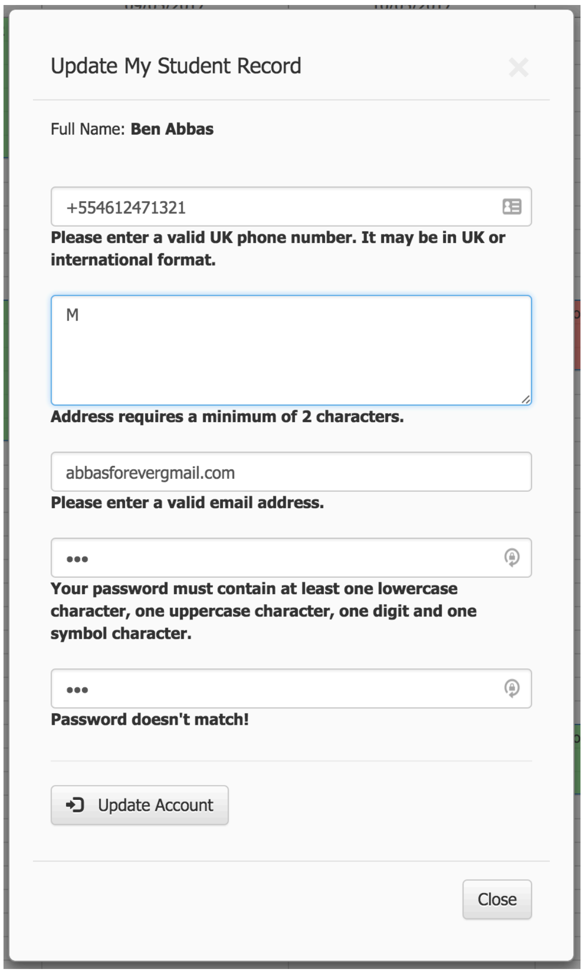
Retype Password

Update Account

Close

The next step in the brief was to create a mechanism by which students can alter their email address, phone number, address and password. To do this, I have used a modal overlay that uses JavaScript, jQuery and AJAX to display a form with some fixed values as well as text fields, which contain dynamic values based on the current user logged in. These values are retrieved by getting the user email or student id from the session and then using this to send via AJAX to carry out PHP that connects to the database and retrieves the student's details and then sends a response in a JSON format containing the user details. As seen on the left, the user details and retrieved and placed in a form.

To validate the form, I have used the jQuery Validate library. This library includes many predefined validations (email, required, length, equalTo, etc.), however, to make sure validation meets the requirements, I have also created RegEx validation method in JavaScript that can be used with jQuery Validate. For example, to meet the password requirement and to ensure a telephone is correct in a UK format, I have used custom RegEx patterns. For example, the following example contains data that does not match the specified rules and thus, validation errors occur that prevent a user from submitting.


Update My Student Record

Full Name: **Ben Abbas**

+554612471321

Please enter a valid UK phone number. It may be in UK or international format.

M

Address requires a minimum of 2 characters.

abbasforevergmail.com

Please enter a valid email address.

•••

Your password must contain at least one lowercase character, one uppercase character, one digit and one symbol character.

•••

Password doesn't match!

Update Account

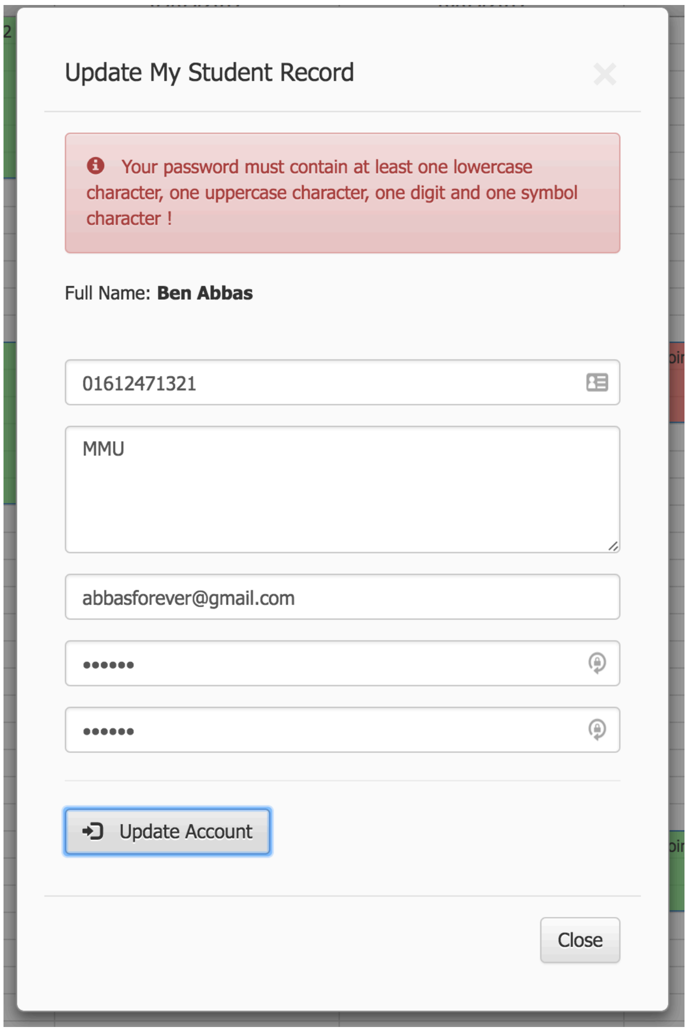
Close

As well as this, I have used the predefined HTML 5.0 function to validate the email if JS were to fail, as well as this, I have used the HTML 5.0 pattern attribute to validate RegEx.

It is also important to have server-side validation, so to do this, I have used PHP to validate errors to prevent un-validated and incorrect data from being added to the database. The PHP contains validation to check for RegEx and check if values are empty. If errors are returned during checks, the error messages will be sent via AJAX to the user where the modal will display the errors and prevent data from being updated within the database. For example, after commenting out jQuery Validate validation in JS, I have been able to test my PHP validation which returns errors through AJAX. On the example on the right, I have used the password 'Hello1'. When the form is submitted, the AJAX has not returned a message to notify success of data update but has returned an error message which is set to the modal following processing.

Test scenarios include the following:

Fields	**Test**
Phone	07965495698 VALID +447965495698 VALID +307965495698 INVALID 02085328105 VALID 01612471321 VALID aword INVALID 000 INVALID 070000000000 INVALID *empty* INVALID
Address	a INVALID 22 VALID *empty* INVALID
Email	*abbasforever@gmail.com VALID* *abbasforevergmail.com INVALID* *abbasforever@gmail. INVALID* *abbasforever@gmail INVALID* *empty* INVALID
Password	Help!mE1 VALID HelpMe INVALID Password1! VALID HelpmE1 INVALID PASSWORD1 INVALID Password1 INVALID Empty VALID
C Password	Matched with Password VALID Not Matched with Password INVALID Empty when Password filled INVALID Empty when Password empty VALID



Your password must contain at least one lowercase character, one uppercase character, one digit and one symbol character !

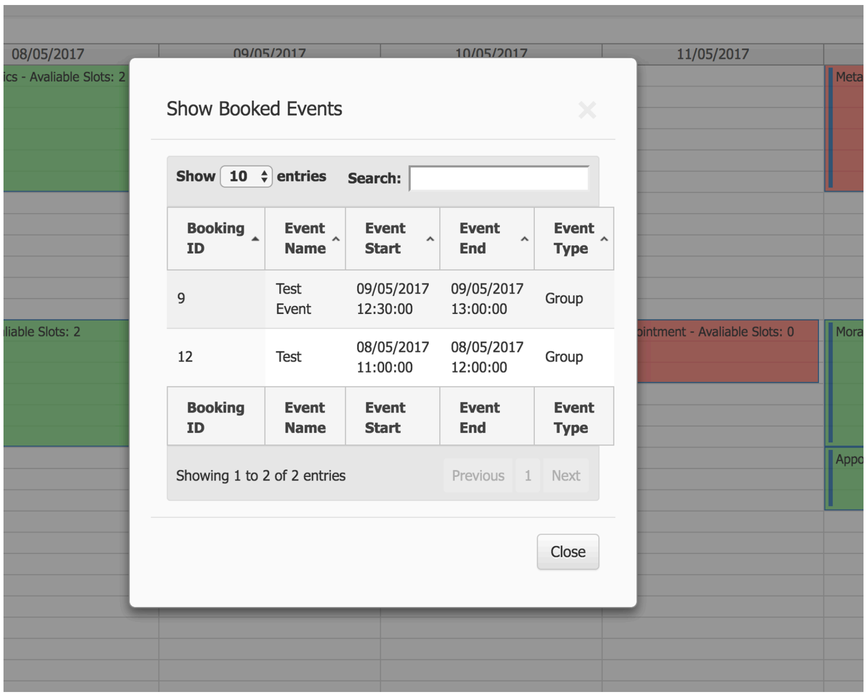
Full Name: **Ben Abbas**

01612471321

MMU

abbasforever@gmail.com

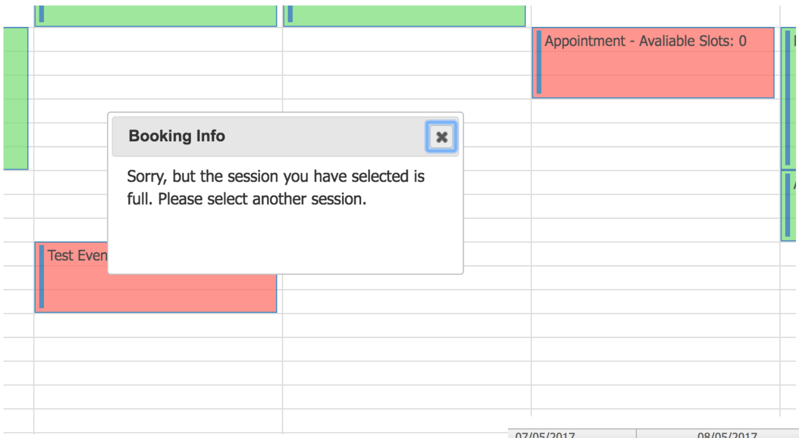
Following successful jQuery data validation, jQuery then 'serializes' the form data and then the data is sent to the PHP via AJAX. The data is then checked here too for validation before allowing the data to enter the database. If there are no errors in the validation, a function will process the data and use SQL update to add the new values to the database.



Booking ID	Event Name	Event Start	Event End	Event Type
9	Test Event	09/05/2017 12:30:00	09/05/2017 13:00:00	Group
12	Test	08/05/2017 11:00:00	08/05/2017 12:00:00	Group

Showing 1 to 2 of 2 entries

As part of section 0.5, task 3, I have used AJAX to retrieve data and jQuery and a dynamic table library based on jQuery UI for the front-end development. When the 'view booked events' menu button is selected in the Bootstrap menu, the booked event data for a specific user will be displayed in a modal.

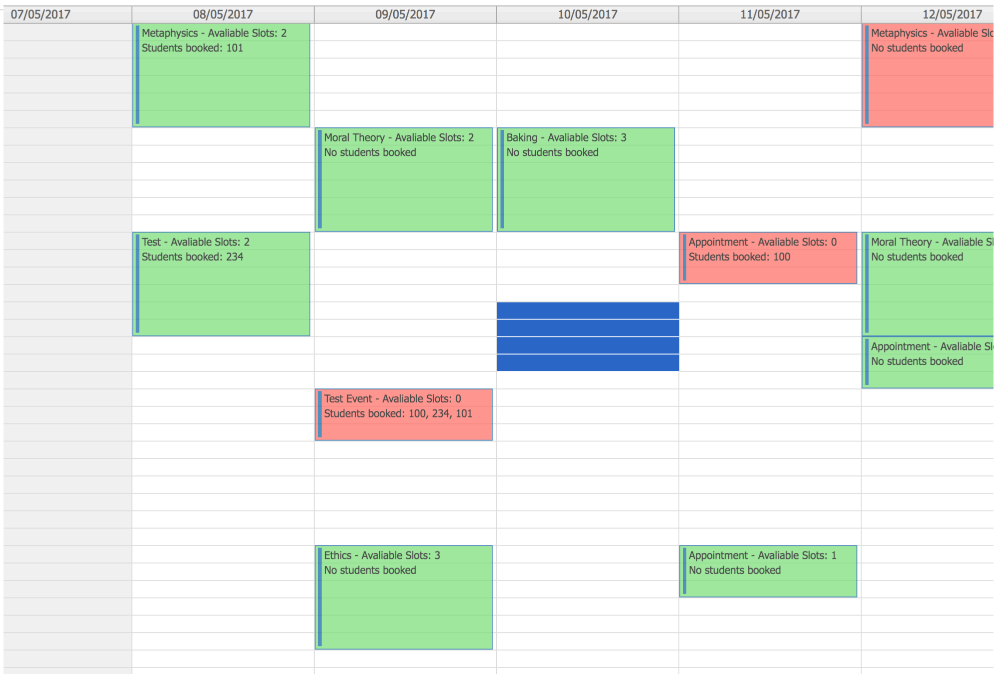

Appointment - Avaliable Slots: 0

Booking Info

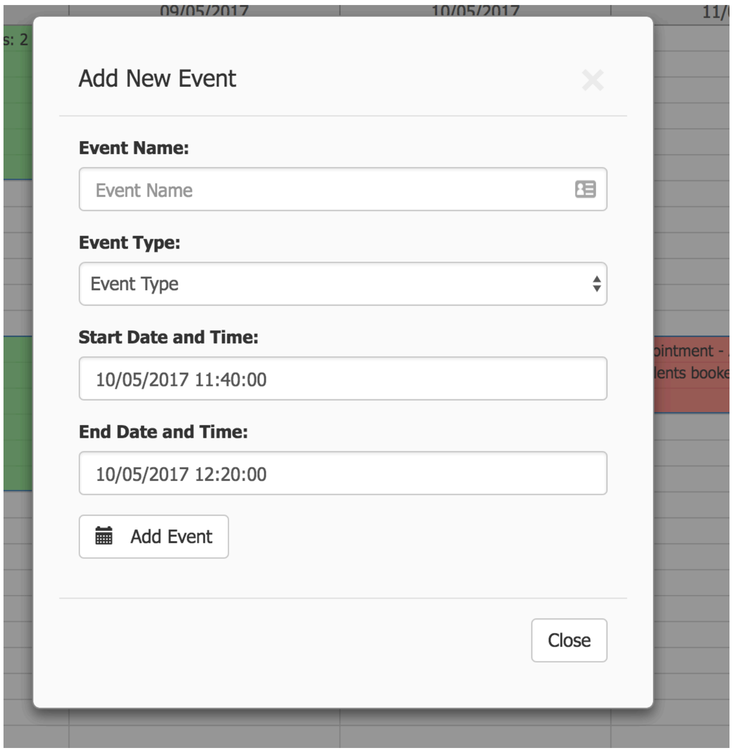
Sorry, but the session you have selected is full. Please select another session.

Test Even

See left; If a user tries to click on an unavailable event on the calendar, the website will display a jQuery UI dialog message, and the user will not be able to book into the event.


07/05/2017	08/05/2017	09/05/2017	10/05/2017	11/05/2017	12/05/2017
	Metaphysics - Avaliable Slots: 2 Students booked: 101	Moral Theory - Avaliable Slots: 2 No students booked	Baking - Avaliable Slots: 3 No students booked	Appointment - Avaliable Slots: 0 Students booked: 100	Metaphysics - Avaliable Slo No students booked
	Test - Avaliable Slots: 2 Students booked: 234	Test Event - Avaliable Slots: 0 Students booked: 100, 234, 101		Appointment - Avaliable Slots: 1 No students booked	Moral Theory - Avaliable Sl No students booked
		Ethics - Avaliable Slots: 3 No students booked			Appointment - Avaliable Sl No students booked

When a time range is selected on the calendar in Socrates view, a modal appears allowing Socrates to add new events. The start and end times will be prefilled based on the time and dates selected. I have used the jQuery UI time picker for these fields to make time and date selection easier for the user. It is also possible to book an event directly from the menu bar at the top of the website. Once the form is completed, AJAX is used to send the data to the PHP file which adds the data into the database. The event type selected in the form will determine whether the value of availability is set to 1 or 3. After this, the modal will close, and the calendar is instantly updated with the new event.


Add New Event

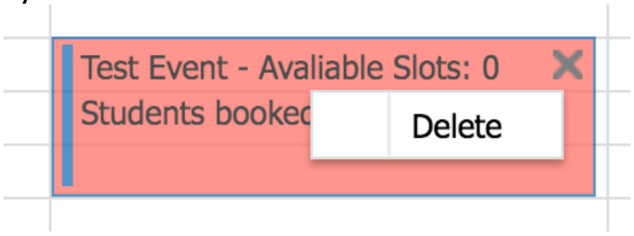
Event Name: Event Name

Event Type: Event Type

Start Date and Time: 10/05/2017 11:40:00

End Date and Time: 10/05/2017 12:20:00

Like the time range selection shown above, it is possible to move and resize events by clicking on events, or the edge of the events and dragging. This triggers AJAX to update the event data in the database. There is also the option to delete via a right-click context menu as well as an 'x' button in the corner of events. Both methods may be used, and both trigger AJAX to remove the events from the database. Of course, these functionalities are all only available within the Socrates view of the system.



This example shows both methods to delete an event.