IFB299 – IT Project Design and Development
Queensland University of Technology
Semester 1, 2018

Personal Portfolio

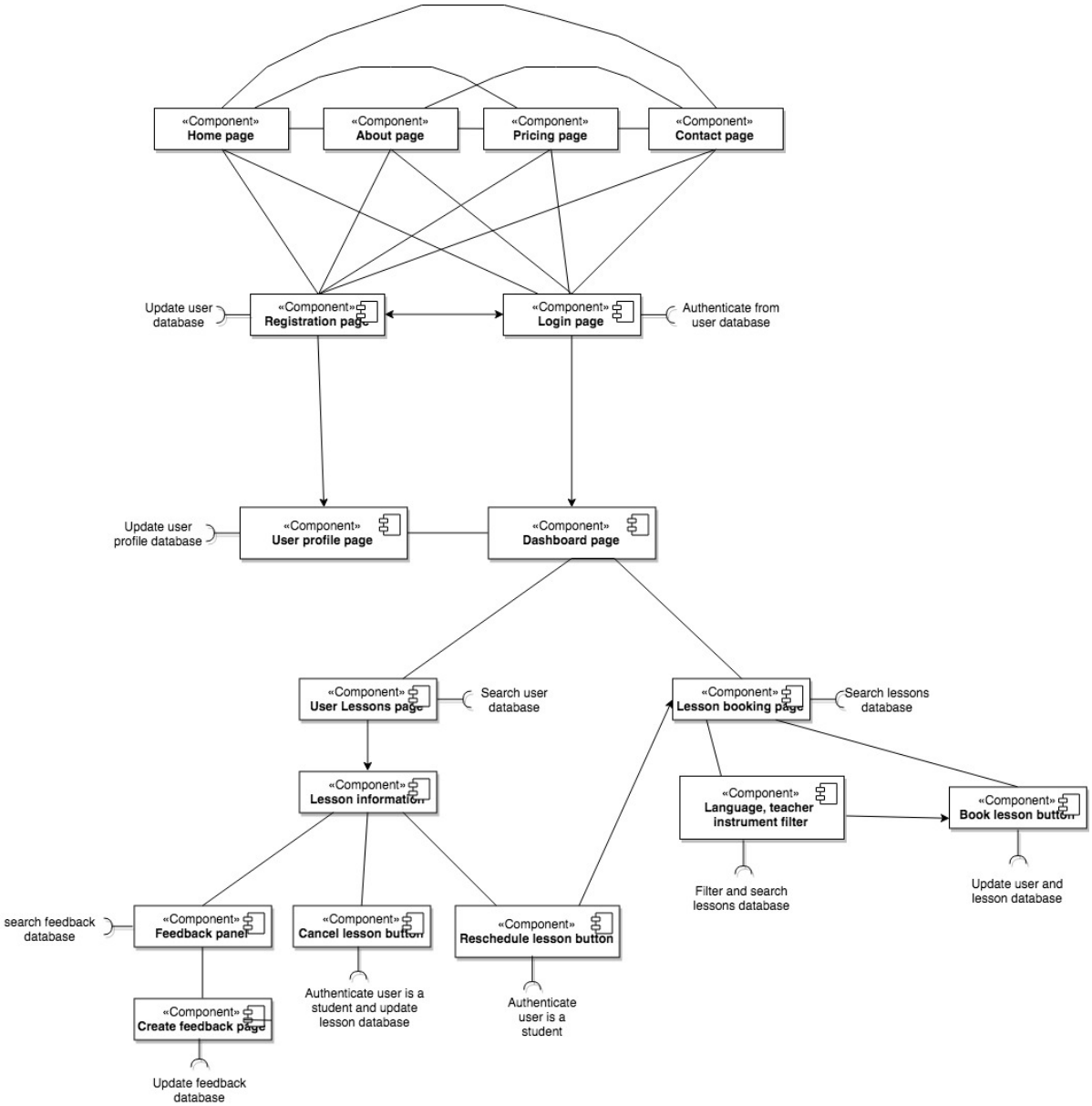
Student: John Santias (N9983244)

Group: 15 – Pink Spoon

BitBucket: https://bitbucket.org/ifb299group15/

Artefact 1 – (Component Diagram)

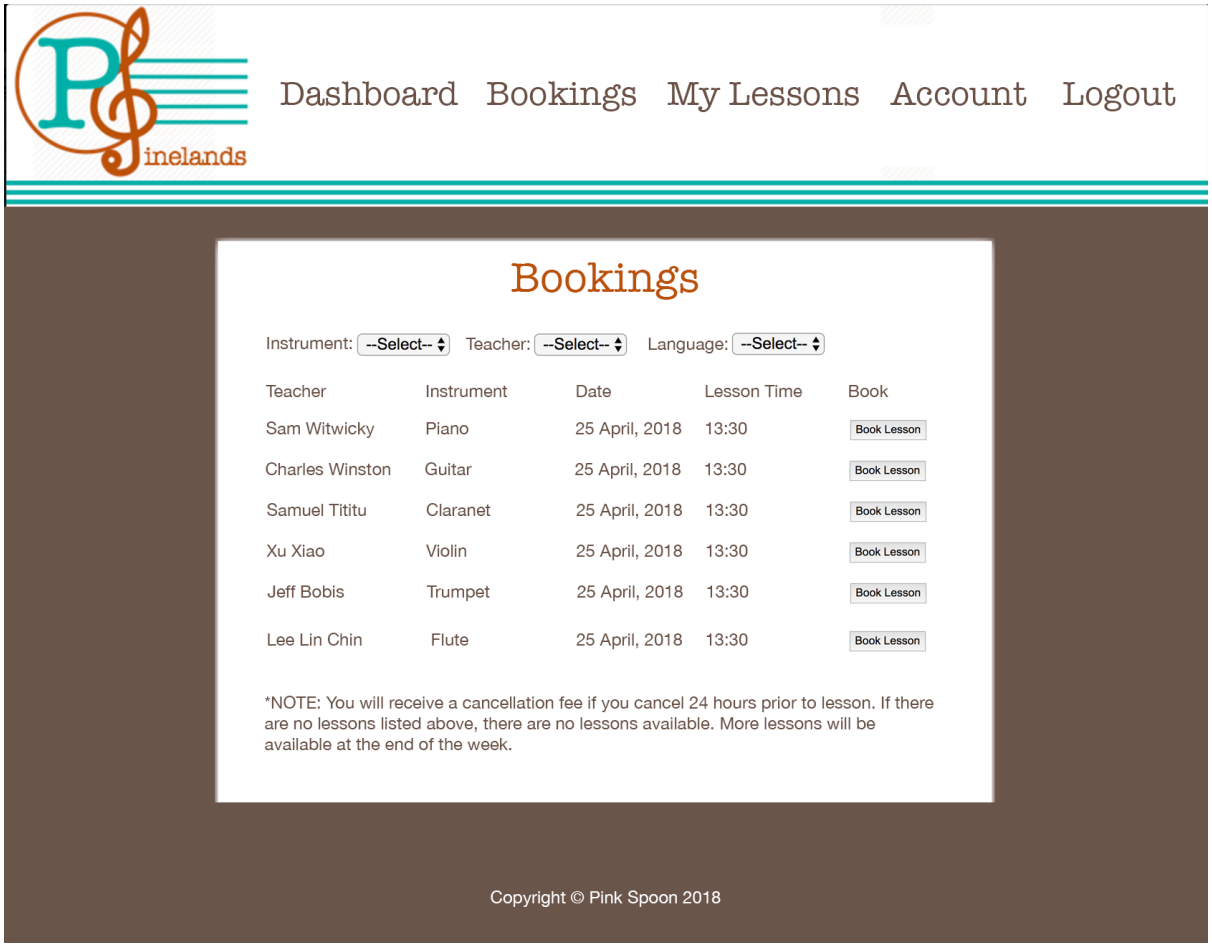
The component diagram describes how our website functions. It communicates with the database to update tables, authenticate users and extract information. This diagram was used as a guidance for the development of the project. We were able to follow the processes and functionality described in this diagram. For example, the user profile page had a submit button to save the user's input by following the component diagram, it specified the website should save those inputs on to the database. So a developer had implemented a function to update the user database when the submit button was pressed. Developers were able to link up the pages together without redirecting the user to an unknown page or causing an error. The diagram saved the developer's time of constantly changing links. Some examples of the website following the component diagrams can be seen in artefact 4 and 5.



Artefact 2 – (UI design)

UI design is the look of a web page focusing on the ease of use and pleasurability for the user. Designs eliminate or fix unwanted features on a web page. Our UI designs allowed the client to confirm what they would want to see on the website and fix or eliminate errors. The client was able to view the designs for the lesson booking, log out and authentication pages. The designs had minor changes to meet the client's expectations. This included moving the filters in one row instead of on top of each other, moving the page title inside the box and centre it. The developers were able to follow the UI designs without having to constantly change/edit the CSS file, which helped them save time from constantly contacting the client for approval.

Bookings page UI High Fidelity

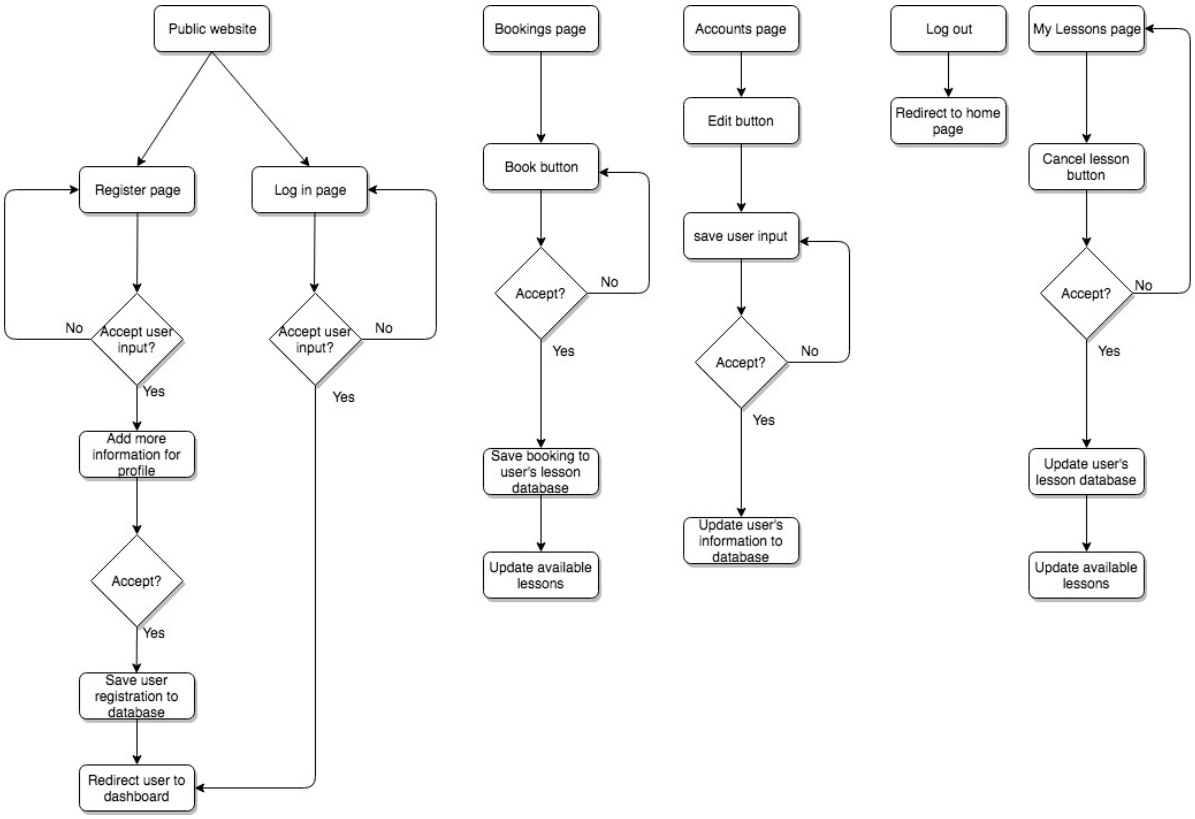


Log out page UI High Fidelity

Unauthorised page UI High Fidelity

Artefact 3 – (Flowchart)

The artefact below are flowcharts for describing how we want to process user requests such as registering a user, process a user's selected booking, edit profiles etc. It shows the structure of the website which is useful for giving a broad overview of the process. For example, the registering user flowchart accepts or denies the user's inputs, then saves it to the database. Flowcharts was a useful tool to remind everyone the processes of each function of the website. It helped developers follow the flow of the process, giving them ideas of how to develop its functionality. Artefacts 4 and 5 follow some of these flowcharts.

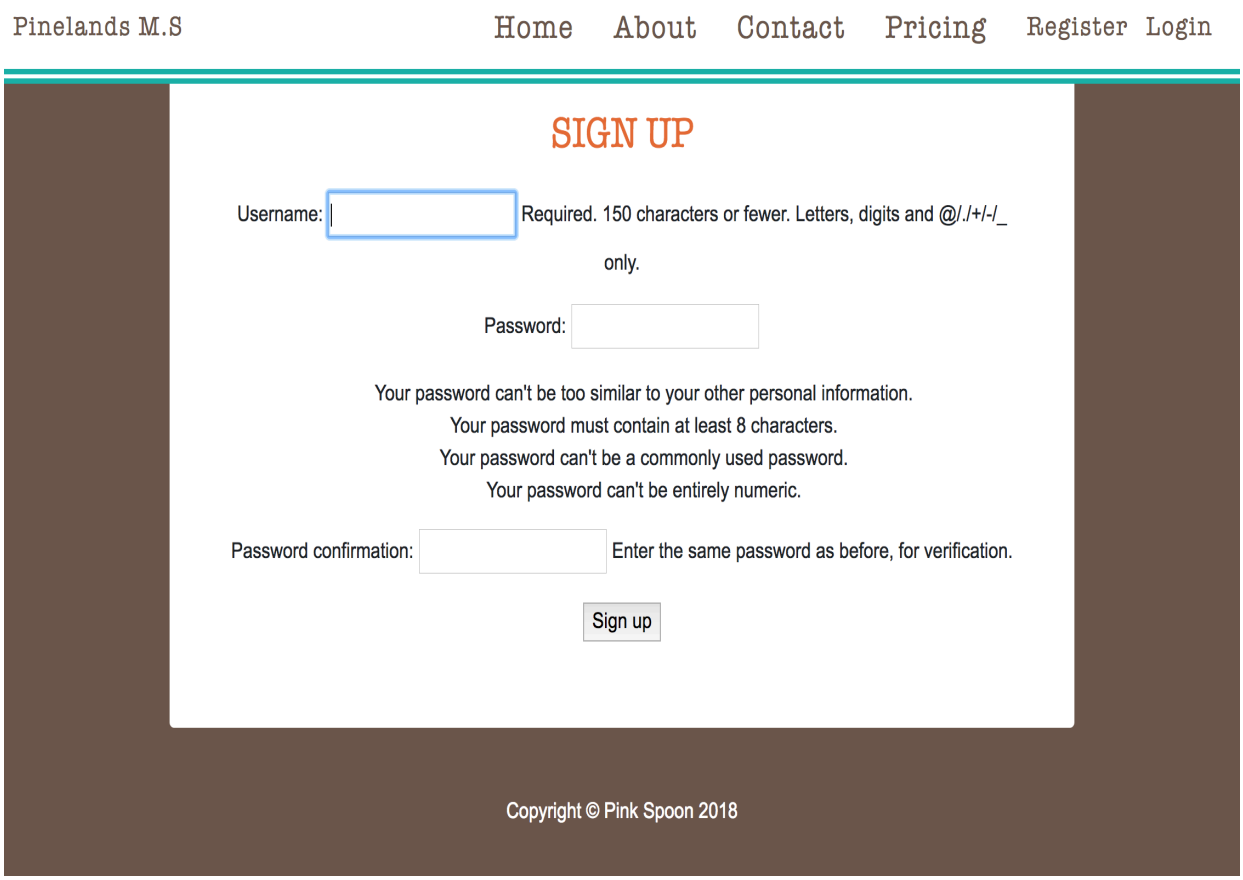


Artefact 4 – (User Accounts)

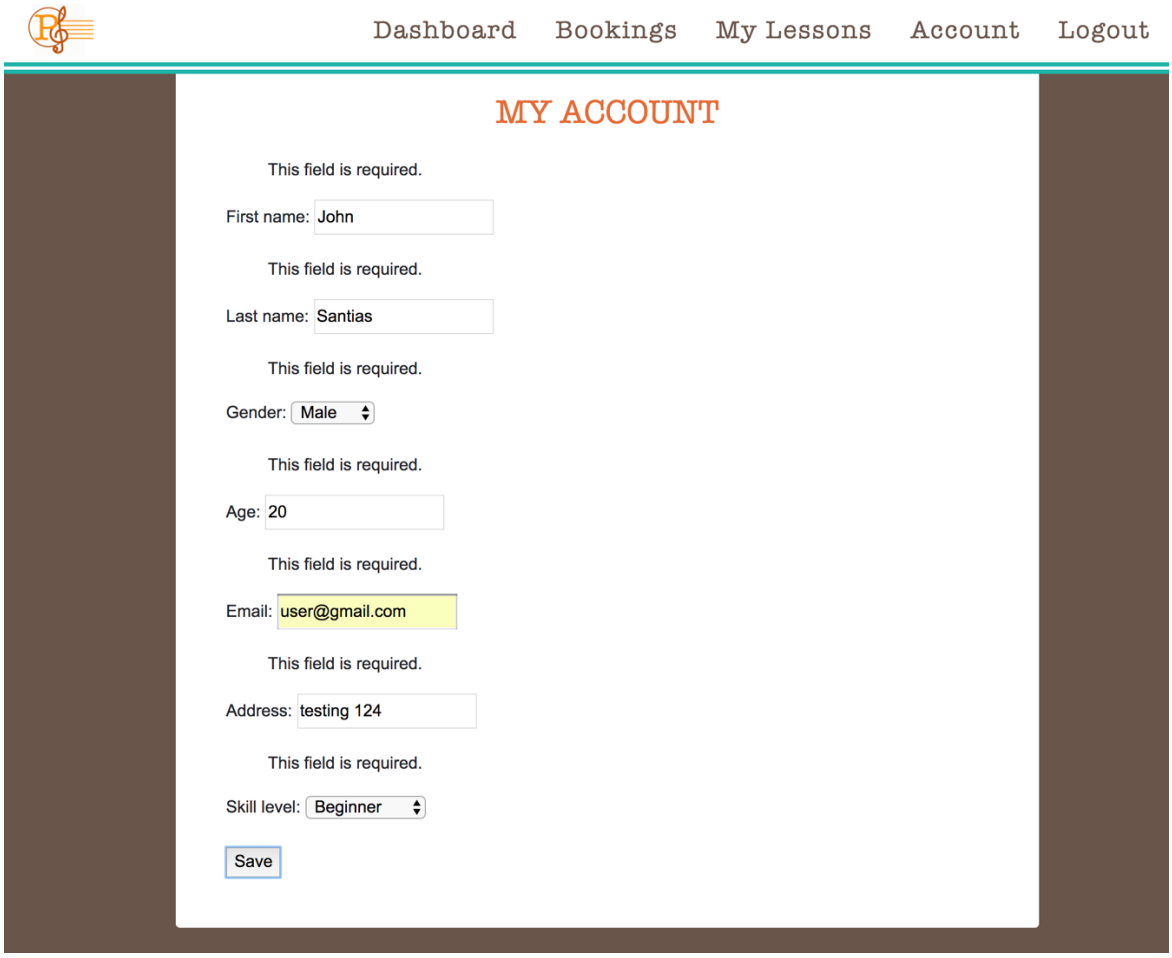
User accounts are an important part of the music school project. Students need an account to access exclusive pages, manage their lessons, view announcements, feedbacks and updates. New customers are able to create an account by clicking on the registration button which will bring up a new form for them to choose a username and password. I was able to implement a registration and login system where the website would register accounts. Whoever, there were some fields that I could not save to the database table, the age, skill level, and gender. James was given the task of trying to get the website to store the missing data into the UserProfile table. He was able to store the inputs into the UserProfile table, but this time the website would not register the user. The solution for this was to separate the two pages, getting the user to register for an account then adding profile information.

Sign up process:

1. User fills in boxes, creates a username and password.



2. User taken to the another page to fill in more details.



Once the user creates the account, Django saves the inputs to the '*auth_user*' database table and then redirects the user to my account page where they will fill out their personal details. Their details are then saved on to a different table called '*music_app_userprofile*'. Their information can be edited by going on to the accounts page at any time.

views.py:

```
def SignUp(request):
    if request.method == 'POST':
        form = UserCreationForm(request.POST) #gets the form
        if form.is_valid():
            user = form.save() #saves form input
            username = form.cleaned_data.get('username') #gets username input
            raw_password = form.cleaned_data.get('password') #gets password input
            auth_login(request, user) #authenticates user and redirects to the next page to add profile information
            return redirect('edit_account')
    else:
        form = UserCreationForm() #displays form
    return render(request, 'signup.html', {'form': form})

@login_required #requires user to be logged in to view this page, otherwise redirected to log in page
def edit_account(request):
    form = SignUpForm(request.POST, instance=request.user) #gets the form
    print(request.POST.get("first_name")) #prints to console for debugging
    user_id = request.user #gets current logged in user
    if form.is_valid():
        existing_instance=UserProfile.objects.filter(user=request.user)
        if (len(existing_instance) == 0): #if user in NOT the user profile table, insert a new row
            new_instance=UserProfile(age=request.POST.get("age"), #insert the row data
            user=request.user, email=request.POST.get("email"),
            skill_level=request.POST.get("skill_level"),
            address=request.POST.get("address"),
            gender=request.POST.get("gender"))
            new_instance.save() #executes sql query to insert row
        else: #if user is NOT in the table
            current_instance=existing_instance[0] #finds user in the userprofile table.
            current_instance.skill_level=request.POST.get("skill_level")
            current_instance.email=request.POST.get("email")
            current_instance.gender=request.POST.get("gender")
            current_instance.address=request.POST.get("address")
            current_instance.age=request.POST.get("age")
            current_instance.save() #executes sql query to update table row
        form.save()
        return redirect('/music_app/account/') #redirects to account page with newly updated information
    return render(request, 'music_app/edit_account.html', {'form' : form})
```

models.py: The *UserProfile* class controls the database *UserProfile* table. The *edit_account* function (from views.py) is using this class to save the user's input to the database.

```
class UserProfile(models.Model):
    SKILLS_CHOICES = [('Beginner','Beginner'), ('Intermediate', 'Intermediate'), ('Expert', 'Expert')]
    GENDER_CHOICES = [('M', 'Male'),('F', 'Female')]
    gender = models.CharField(choices=GENDER_CHOICES, max_length=1, blank=False)
    age = models.IntegerField(null=False)
    email = models.CharField(max_length=250)
    address = models.CharField(max_length=250)
    skill_level = models.CharField(choices=SKILLS_CHOICES,max_length=12, blank=False)
    user = models.OneToOneField(User, on_delete=models.CASCADE, null=False, related_name='profile')

    def __str__(self):
        return self.user.username
```

forms.py: The *SignUpForm* class below displays a form on the 'my Account' page. The form is using the *UserProfile* model which will provide the fields specified in models.py

```
class SignUpForm(forms.ModelForm):
    first_name = forms.CharField(max_length=100, required=True)
    last_name = forms.CharField(max_length=100, required=True)
    gender = forms.ChoiceField(choices=GENDER_CHOICES)
    age = forms.IntegerField(max_value=120, required=True)
    email = forms.EmailField(required=True)
    address = forms.CharField(max_length=250, required=True)
    skill_level = forms.ChoiceField(choices=SKILLS_CHOICES)

    class Meta:
        model = UserProfile
        fields = (
            'first_name',
            'last_name',
            'gender',
            'age',
            'email',
            'address',
            'skill_level',
            )
```

auth_user table:

id	password	last_login	is_superuser	username	first_name	last_name	email	is_staff	is_active	date_joined
1	pbkdf2_sha256$100000$47PTalbUvQc5$3jCTU…	NULL	0	john				0	1	2018-05-02 20:18:29.661251
2	pbkdf2_sha256$100000$2Yi3SnrmgSbn$EBEt…	2018-05-02 20:20:22.007411	0	john345	helo	santias	user@gmail.com	0	1	2018-05-02 20:20:21.844079
3	pbkdf2_sha256$100000$T2tf95UbB9R8$+V5kx…	2018-05-02 20:37:39.699825	0	testing123	john	san	user@gmail.com	0	1	2018-05-02 20:37:39.502823
4	pbkdf2_sha256$100000$Ocg91Ui964mz$afiHb…	2018-05-02 21:11:46.712731	0	afv	dghn	fgb	16@gma.lo	0	1	2018-05-02 21:11:46.566324
5	pbkdf2_sha256$100000$NWEPRbfiaxHI$XgCj…	2018-05-03 02:09:45.039554	0	testing12309	testing1324	hello	user@gmail.com	0	1	2018-05-03 02:09:44.794842
6	pbkdf2_sha256$100000$jJZ85ol6yhAL$k9vYtG…	2018-05-03 02:36:21.489555	0	newuser	John	Liam	user@gmail.com	0	1	2018-05-03 02:36:21.207235
7	pbkdf2_sha256$100000$jksMxkwOJhnV$BUj8…	2018-05-03 07:01:49.019482	0	johnsant	John	Santias	user@gmail.com	0	1	2018-05-03 07:01:48.869371

userprofile table:

id	gender	age	address	skill_level	user_id	email
1	M	39	49 sotk	Beginner	5	user@gmail.com
3	M	19	user te…	Expert	6	user@gmail.com
4	M	20	testing…	Beginner	7	user@gmail.com
NULL	NULL	NULL	NULL	NULL	NULL	NULL

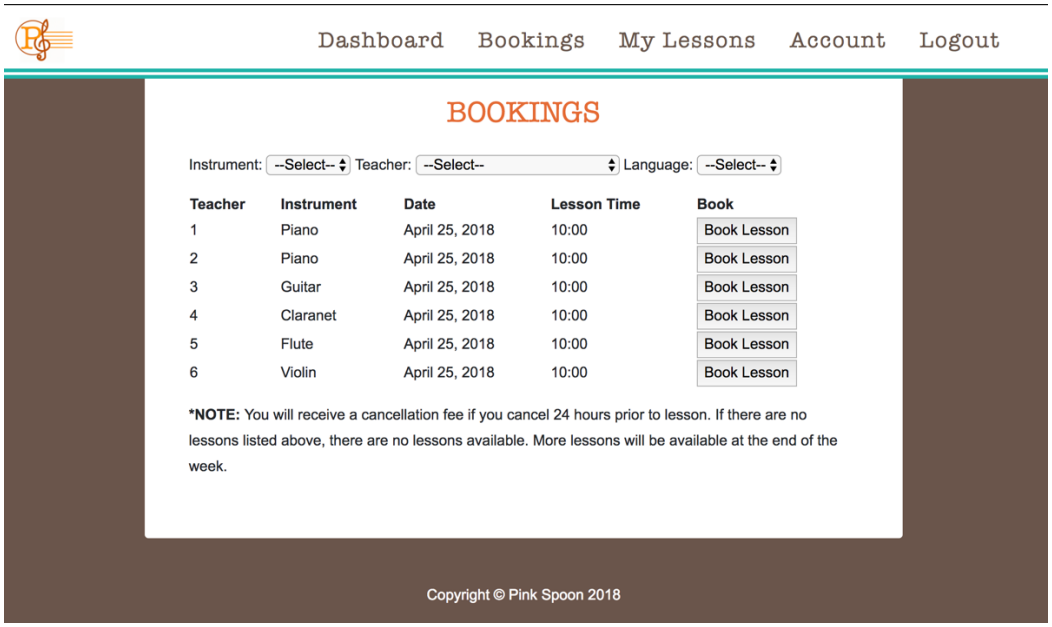
The accounts system will be very useful as it can allow students to manage their lessons, their accounts, hire instruments, pay for lessons etc. New customers are also able to register and be part of the music school community. Some pages such as the bookings, my account, dashboard, my lessons require the user to be authenticated before displaying the page. The website now allows administrators to manage user accounts.

Artefact 5 – (Bookings System)

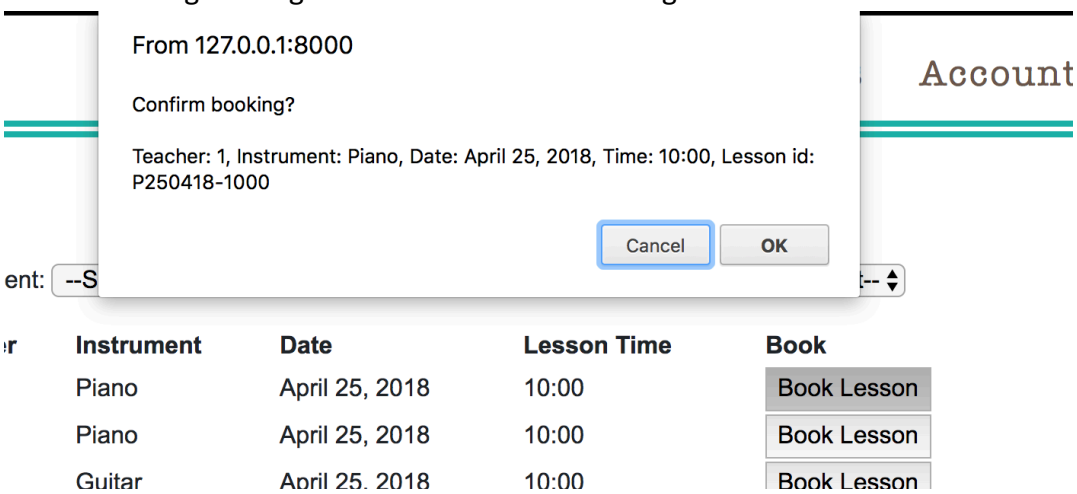
The booking system allows students to select which kind of lessons they want to go to. The page provides a whole list of lessons for the following two weeks. Lessons are available from 10 am to 6 pm every day with different instruments to learn. With two clicks, lessons are immediately assigned to the user. This system is a major part of the project, as lessons are now more organised and no one has to call in or see someone for a lesson. The bookings system is one of the main features. I was able to extract lesson data from the database and display it on the page inside a table format. JavaScript was also included to confirm the user's booking. Once booking is confirmed, the lesson is hidden from the page.

Bookings a lesson:

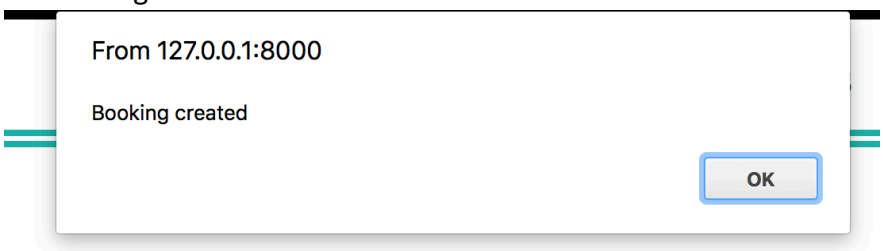
1. Booking the lesson listed on top



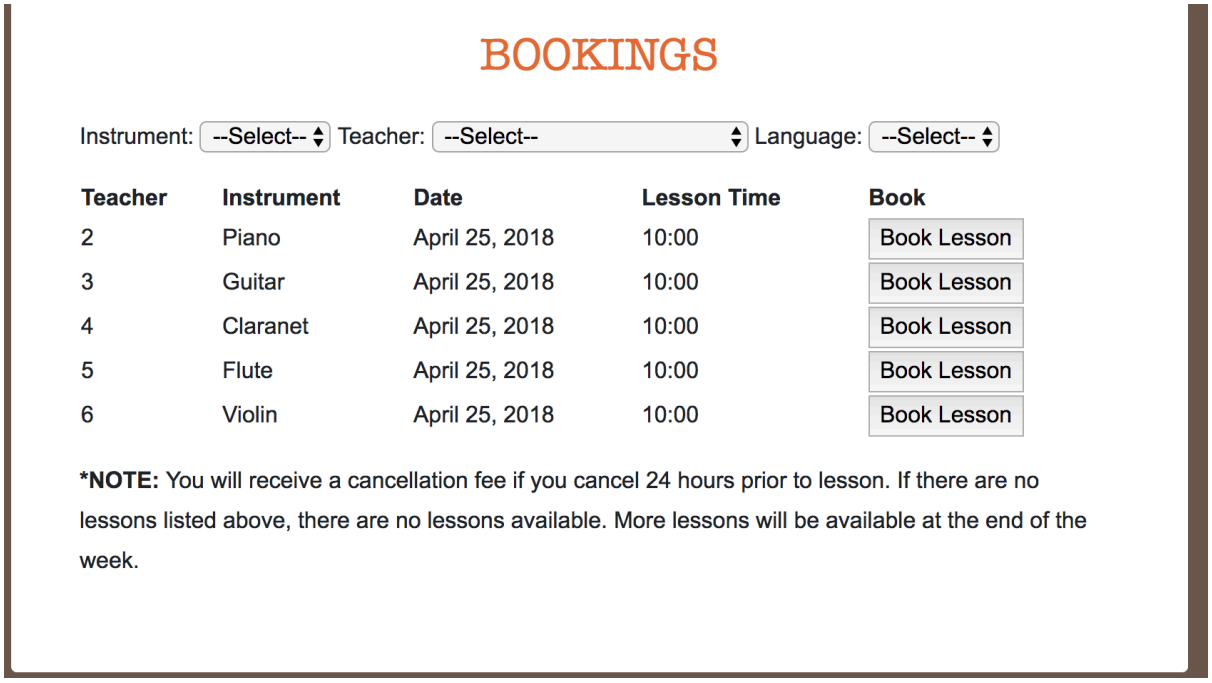
2. Message asking for confirmation on booking.



3. Booking is confirmed and saved.



4. Booked lesson disappears



5. Table is updated

id	Instrument	Date	Time	Language	teacher_id	Lesson_id	Booked
1	Piano	2018…	10:0…	English	1	P250418-…	NO
2	Piano	2018…	10:0…	English	2	P250418-…	NO
3	Guitar	2018…	10:0…	English	3	G250418…	NO
4	Claranet	2018…	10:0…	English	4	C250418…	NO
5	Flute	2018…	10:0…	English	5	F250418-…	NO
6	Violin	2018…	10:0…	English	6	V250418-…	NO
NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL

id	Instrument	Date	Time	Language	teacher_id	Lesson_id	Booked
1	Piano	2018…	10:0…	English	1	P250418-…	YES
2	Piano	2018…	10:0…	English	2	P250418-…	NO
3	Guitar	2018…	10:0…	English	3	G250418…	NO
4	Claranet	2018…	10:0…	English	4	C250418…	NO
5	Flute	2018…	10:0…	English	5	F250418-…	NO
6	Violin	2018…	10:0…	English	6	V250418-…	NO
NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL

bookings.html

```
    <table style="width: 100%">
      <tr>
        <th>Teacher</th>
        <th>Instrument</th>
        <th>Date</th>
        <th>Lesson Time</th>
        <th>Book</th>
      </tr>
      <!-- Gets each row from the schedule table  -->
      {% for item in data_list %}
      <tr>
        <td>{{item.teacher_id}}</td>  <!-- Gets the teacher column data of the row -->
        <td>{{item.Instrument}}</td> <!-- Gets the instrument column data of the row -->
        <td>{{item.Date}}</td> <!-- Gets the date column data of the row -->
        <td>{{item.Time|time:"H:i"}}</td> <!-- Gets the teacher column data of the row -->

        <!-- Gives button value to display in the javascript confirm box -->
        <td><button name="checks" id="{{item.id}}" value="Teacher: {{item.teacher_id}}, Instrument: {{item.Instrument}}, Date: {{item.Date}}, Time:
        {{item.Time|time:'H:i'}}, Lesson id: {{item.Lesson_id}}" onclick="clicked(this.value, this.id)">Book Lesson</button></td>
      </tr>
      {% endfor %} <!-- Ends the if statement the teacher column data of the row -->
    </table>
  </p>
  <span>
    <p><strong>*NOTE:</strong> You will receive a cancellation fee if you cancel 24 hours prior to lesson. If there are no lessons listed above, there are no
    lessons available. More lessons will be available at the end of the week.</p>
    </p>
  </span>
  <script type="text/javascript">
  function clicked(value, id) {
     if (confirm('Confirm booking? \n \n' + value)) { //confirm box pops up when the user clicks a book button and displays button's value
        $.ajax({'type': 'POST', //type of request when calling the bookings function in views.py
                'url': '/music_app/bookings/', //calls the view.py bookings function to execute the else statement
                'data': {'id': id}, //sends id data to the bookings function
                'success': function(resp) {
                      if(resp.status == 'ok') {
                        alert('Booking created'); //successful booking message
                      }
                      else {
                        alert(resp.message); //error, invalid lesson or doesn't exist
                      }
                }
               });
      location.reload(); //reloads page
```

views.py

```
@csrf_exempt #disables csrf protection
def bookings(request):
    if (request.method == 'GET'):
        data_list = schedule.objects.filter(Booked="NO") #Shows lessons that are available
        context = {'data_list':data_list } #gets all rows in the schedule table
        return render(request, 'music_app/bookings.html', context) #displays html page

    else:

        try:
            schedule_id = int(request.POST.get('id')) #Obtains the button's id (the one that was clicked)
            sched = schedule.objects.get(id = schedule_id) #finds row containing the schedule it
            booking, created = Bookings.objects.get_or_create(schedule=sched, student=request.user) #inserts a new row in the bookings table, inserts the user and schedule id
            schedule.objects.filter(id=schedule_id).update(Booked='YES') #updates the row containing the schedule id and changes its booked status from 'NO' to 'YES'
            return JsonResponse({'status': 'ok'}) #success message
        except schedule.DoesNotExist:
            return JsonResponse({'status':'error', 'message': 'Schedule does not exists'}) #returns error message
        except ValueError:
            import traceback
            traceback.print_exc()
            return JsonResponse({'status':'error', 'message': 'Invalid shcedule id'})#returns error message
```

models.py: The *schedule* class controls the database *schedule* table. Same for the bookings class. The *bookings* function (from views.py) is using the bookings class to save the lesson to the bookings database table and it uses the schedule class to find lesson id and update the booking status.

```
class schedule(models.Model):
    teacher = models.ForeignKey(teacher, on_delete=models.CASCADE)
    Instrument = models.CharField(max_length=100)
    Date = models.DateField()
    Time = models.TimeField(auto_now=False, auto_now_add=False)
    Lesson_id = models.CharField(max_length=100, null=False)
    Language = models.CharField(max_length=100)
    Booked = models.CharField(max_length=100)

class Bookings(models.Model): #shows student's bookings
    student = models.ForeignKey(User, on_delete=models.CASCADE)
    schedule = models.ForeignKey(schedule, on_delete=models.CASCADE)
```