

***OCR A-Level Computer Science Programming Project***

est

**SoundBro.**

2023

m u s i c   e x a m   s c h e d u l i n g

# **SoundBro: Music Exam Scheduling Software**

*By Rohan Desai*

<b>1 Analysis.....</b>	<b>2</b>
1.1 Problem Analysis.....	2
1.2 Suitability of computational methods.....	9
1.3 Stakeholders.....	11
1.4 Research of existing solutions.....	16
1.5 Features of the proposed solution.....	25
1.6 Hardware and software requirements.....	31
1.7 Success criteria.....	32
<b>2 Design.....</b>	<b>37</b>
2.1 Problem decomposition.....	37
2.2 Structure of solution.....	39
2.3 Usability features.....	63
2.4 Data and Validation.....	69
2.5 Algorithms.....	87
2.6 Iterative test data.....	134
2.7 Post-development test data.....	140
<b>3 Development.....</b>	<b>145</b>
3.1 Database setup.....	145
3.2 Login functionality.....	157
3.3 Registration feature.....	173
3.4 nextBreak function.....	195
3.5 meetingRegulations function.....	202
3.6 addBreak function.....	214
3.7 accompAvailable function.....	221
3.8 recalculateOrder function.....	237
3.9 The schedule function.....	273
3.10 System integration.....	310
3.11 Gathering inputs.....	325
3.12 Publishing the schedule.....	335
<b>4 Post Development Testing.....</b>	<b>354</b>
4.1 Interview with Ms Pearcey.....	354
4.2 Functionality testing.....	364
4.3 (R) Robustness testing.....	377
4.4 Usability testing.....	383
<b>5 Evaluation.....</b>	<b>389</b>
5.1 Meeting the success criteria.....	389
5.2 Meeting the usability features.....	395
5.3 Further development.....	406
5.4 Limitations.....	412
5.5 Maintenance.....	415
<b>6 Appendix.....</b>	<b>417</b>
6.1 Full stakeholder responses (conducted during analysis).....	417
6.2 Post-development Ms Pearcey interview transcript.....	424
6.3 References.....	436
6.4 Final code.....	437

# 1 Analysis

## 1.1 Problem Analysis

### 1.1.1 Introduction

Examinations for musical grades take place in schools across the country. Usually, these exams will take place over a day, or a few days. As a keen drummer who has achieved grade 8, I have personal experience of this system and have found it frustrating to have changes in my exam time, this leads to stress as well as decreased performance. Students taking these exams are already under stress, and additional confusion about examination time can be detrimental to their performance.

Exam scheduling can become a very complex process. For these exams to run smoothly and efficiently, they must be carefully scheduled according to the availability of all parties involved. Furthermore, different exams have different requirements and time constraints. Schools must follow strict deadlines and regulations from unhelpful exam boards, while also ensuring accompanist time is not wasted. Students also tend to be late to these deadlines or may input their information incorrectly due to carelessness. This often means that undisclosed school trips/medical appointments come in the way of their musical career progression, and cause more of the examiner's time to be wasted.

The bewildering regulations of exam boards also pose significant problems to the organiser. After gathering availability for students and other stakeholders and processing that to create a schedule, that same schedule must be inputted into the exam board's system to conform to their standard for breaks. From both my and my client's understanding, these regulations make little to no sense, further limiting the chance that all parties are kept happy.

After a schedule is created, the organiser must produce a printout manually and display it on a notice board, and complete the tedious task of emailing all those involved. School students tend to be less mature and responsible as adults, and often turn up late, or sometimes not at all, as they forget about the exam. Figure 1.1.1a clearly shows the stages involved in this process.

It is evident that the scheduling of music exams to maximise efficiency can become a very complex process, which is why I have decided to find a solution to this issue for my school, as well as others across the country.

**Please note:** If not already familiar, an **accompanist** plays along with a student during a music exam, if the piece requires an accompaniment. They may be music teachers, and one person can accompany multiple students. Each accompanist will have a certain availability that the schedule must adhere to. This term is used frequently throughout the document.

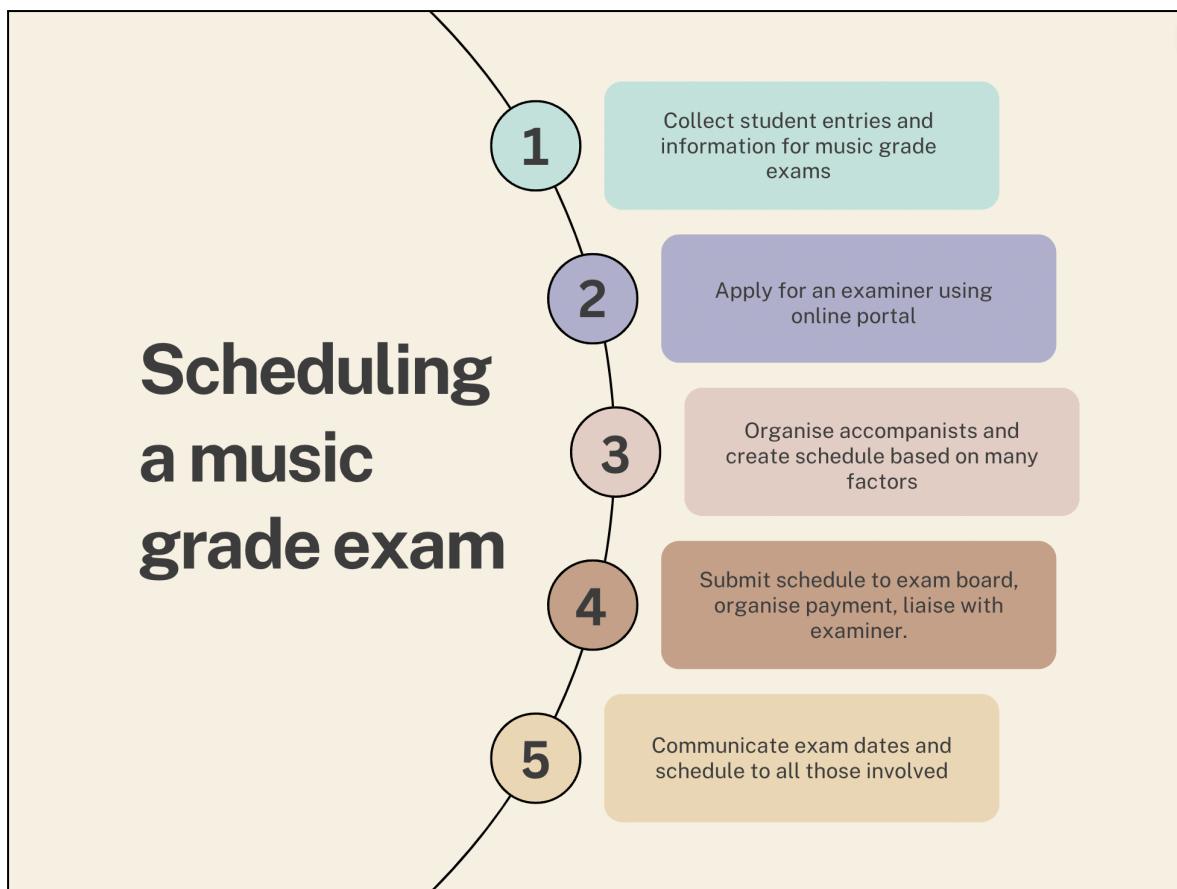


Figure 1.1.1a: A brief overview of the scheduling process (made with Canva)

### 1.1.2 TCL Regulations

Trinity College London is the exam board primarily used by my school for music grade examinations. It has 2 main divisions: classical and jazz, and rock and pop. These divisions can largely be ignored, as they make minimal difference to the scheduling process. The only notable difference is that the divisions have specified different exam lengths for their grades, as shown in figure 1.1.2b, so this must be programmed into my solution. As my project is to schedule TCL exams only, only their regulations will be integrated into my solution.

## MUSIC TIMETABLES

A standard start time is anywhere between 9:00 and 10:00am. The examiner will arrive 30 minutes before the first exam.

An examiner may work for no more than 2 hours without a 15 minute break. A one hour lunch break should be timetabled after no more than 3.5 hours. The maximum exam time allowed per day (not including breaks) is 6.5 hours. For a full day of exams, a one hour lunch break and 15 minute breaks in mid-morning and mid-afternoon should be included. This is illustrated in the following example:

### SAMPLE TIMETABLE

EXAMS	BREAKS
09:00-11:00	11:00-11:15
11:15-12:45	12:45-13:45 (lunch)
13:45-15:15	15:15-15:30
15:30-17:00	

The total number of hours an examiner spends at the centre in one day should not exceed 8 hours. This includes both examining time and breaks. Deviations from this standard must be cleared with Trinity no less than four weeks in advance of the exam date.

Timetable the exams so that they are grouped by syllabus/instrument family. Up to date timings for each music exam are shown on the next page and are included in the appropriate syllabus. Extra time may be required for SEND candidates as advised by Trinity's SEND team [trinitycollege.com/music-csn](http://trinitycollege.com/music-csn). This should be added to the timetable as required.

When examiners change venue during the day, for instance when visiting a school or moving room for the afternoon, the necessary packing/unpacking and travel time must be allowed for and not taken out of the examiner's breaks.

Figure 1.1.2a: Rules for timetabling exams (<http://www.trinitycollege.co.uk/MusicTimetabling>)

There are several regulations which the organiser must follow when creating a timetable, as shown in figure 1.1.2a. Below I have summarised the regulations:

- Start time is from 9am to 10am
- Each session has a maximum duration of 2 hours
- After 3.5 hours (not including breaks), a lunch break must be scheduled
- Lunch break is 1 hour
- There must be two 15 minute breaks, one in the mid-morning and one in the mid-afternoon
- Maximum exam time is 6.5 hours per day (not including breaks)
- Maximum overall time is 8 hours per day
- Exams should be grouped by syllabus/instrument family (they should be grouped by the categories shown in figure 1.1.2b and below)

Additionally, the duration of each exam depends on the instrument, grade, and division (TCL classical and jazz / TCL rock and pop). These are shown in figure 1.1.2b. Please note that I

am only catering my solution for initial grade to grade 8, not diplomas or group certificate exams. All TCL instruments within the groups shown will be supported.

LEVEL	EXAM TIMINGS (MINUTES)				
	Piano	Drum kit and percussion	Harp Organ Rock & Pop exams Electronic keyboard	Brass Guitar Strings Singing Woodwind	
GRADE EXAMS					
Initial	10	13	13	11	
Grade 1	11	15	13	13	
Grade 2	11	15	15	13	
Grade 3	12	16	15	13	
Grade 4	16	21	20	18	
Grade 5	16	21	20	18	
Grade 6	22	27	25	23	
Grade 7	22	27	25	23	
Grade 8	27	32	30	28	

Figure 1.1.2b: Duration of each grade exam (<http://www.trinitycollege.co.uk/MusicTimetabling>)

Each instrument family in the table can be split into individual instruments, as shown below:

- **Piano**
  - Piano
- **Drum kit and percussion**
  - Drum kit
  - Tuned percussion
  - Snare drum
  - Timpani
  - Orchestral percussion
- **Harp**
  - Pedal harp
  - Non-pedal harp
- **Organ**
  - Organ
- **Rock and Pop exams**
  - Rock and Pop Bass
  - Rock and Pop Drums
  - Rock and Pop Guitar
  - Rock and Pop Keyboards
  - Rock and Pop Vocals
- **Electronic keyboard**
  - Electronic keyboard
- **Brass**
  - French horn

- Eb tenor horn
- Trumpet
- Cornet
- Flugelhorn
- Eb soprano cornet
- Euphonium
- Baritone
- Trombone
- Bass trombone
- Tuba
- Eb bass
- Bb bass

● **Guitar**

- Acoustic guitar
- Classical guitar

● **Strings**

- Violin
- Viola
- Cello
- Double bass
- Scottish traditional fiddle

● **Singing**

- Singing

● **Woodwind**

- Flute
- Clarinet
- Oboe
- Bassoon
- Saxophone
- Recorder
- Jazz flute
- Jazz clarinet
- Jazz saxophone
- Accordion

These categories have been expanded after clarification from the official TCL website (<https://www.trinitycollege.com/qualifications/music/music-certificate-exams>). I aim to cater for every single instrument shown above. The schedule should aim to schedule exams so they are grouped into these categories.

On top of all of these regulations, I must make sure the timetable fits within the availability of all accompanists. To save their time, each accompanist's exams should be grouped together as much as possible. The timetable also has to fit in all students who make an entry to take an exam.

### 1.1.3 Updated TCL regulations

**UPDATE: I have now decided to focus this project on my school only (see [1.5.5 Limitations](#)). As a result, my client (Ms Pearcey) has specified some additional requirements for this project.**

Ms Pearcey and I talked frequently after I made the decision to schedule exams for this school only, not for all schools across the country. This has led to more specific requirements, and a clearer understanding of what scheduling may actually involve. After communication in person and via email, I have summarised all regulations again, with all changes.

- 9am start time, 5pm end time
- Roughly 10:45 break
- Roughly 12:30 lunch
- Roughly 15:15 break
- Lunch is 1 hour exactly
- Each break is 15 mins exactly
- Each session has a maximum duration of 2 hours
- After 3.5 hours (not including breaks), a lunch break must be scheduled
- Maximum exam time is 6.5 hours per day (not including breaks)
- Maximum overall time is 8 hours per day
- Exams should roughly follow the schedule as shown below (this meets all above regulations)

Time (roughly)	Activity	Duration
09:00 - 10:45	Exams	105 (approx)
<b>10:45 - 11:00</b>	<b>Break</b>	<b>15</b>
11:00 - 12:30	Exams	90 (approx)
<b>12:30 - 13:30</b>	<b>Lunch</b>	<b>60</b>
13:30 - 15:15	Exams	105 (approx)
<b>15:15 - 15:30</b>	<b>Break</b>	<b>15</b>
15:30 - 17:00	Exams	90 (approx)

Figure 1.1.3a: Rough exam schedule

- Each accompanist will have their own availability. The schedule must work with their availability.
- Scheduling all examinees must be attempted
- The duration of each exam depends on the instrument, grade and division (see [1.1.2](#) for more details)
- The schedule should try to group exams in the following order:

- Grouped by accompanist
- Grouped by instrument family
- Grouped by instrument
- Sorted by grade (ascending)

This means that within each accompanist ‘block’, the exams should be sorted into their instrument families, and within that they should be sorted into their instruments, moving up by grade. Obviously, this is just to make the process more efficient. The other rules above are all strict, and must be followed.

By scheduling exams to cater exactly to Ms Pearcey’s needs, I hope to reduce her workload further and maximise the efficiency of the produced timetable. I aim to meet her requirements through regular communication and feedback during every stage of the process.

## *1.2 Suitability of computational methods*

### *1.2.1 Why use a computer?*

Student and accompanist information is already being stored in a spreadsheet by my client, but using databases to store this information is the most effective, allowing emails, availability and other factors to be recorded accurately. Databases can easily be understood and processed by a computer program, making a software solution suitable for this problem.

The central issue of this project is creating a schedule that is the most efficient for all parties. As well as balancing accompanist availability with the TCL exam board regulations, Ms Pearcey must schedule breaks correctly and group accompanists together to save time. She also has to deal with any issues raised by students. Managing all of these factors becomes quite difficult, as they combine to make quite a daunting task. However, a computer can store a large number of variables and make complex decisions using algorithms to consistently give accurate results. With the right algorithm, something that takes Ms Pearcey several days can be carried out in a matter of minutes.

Another monotonous task is creating a printout to put on the notice boards. For each exam day, the layout will be the same, but the schedule itself will differ. For a program that has created this schedule, automating this process will be simple but very effective, saving further time. This is because all the key information has already been inputted into the system, the program will just have to utilise it correctly to create a printout.

By using a computer-based solution, schedules can be communicated immediately to all relevant people. This gives everyone more time to raise any concerns, so the scheduling process as a whole becomes more efficient. In general, without a computer, tasks become less efficient, less accurate and more difficult to complete.

### *1.2.2 Computational methods*

Abstraction can be used to remove unnecessary detail from a problem, making it easier to solve. This method can be used throughout the project, for example when collecting data. It is important for databases to be organised effectively to prevent redundant and inconsistent data, which can introduce errors in the program. Therefore, not all data is collected, only those that are necessary for solving the problem. This includes the student's year group, student's form group, parent's name, the accompanist's instrument etc. By focussing on essential parts of this system, I can ensure that the project does not deviate from its original goal.

Another use of abstraction in this project is when displaying specific information to users. They are prevented from viewing the entire schedule, as this involves unnecessary data which can make the user feel confused and increase the chance of misunderstanding the time slot. Therefore, data is abstracted and only the time slots relevant to each user is displayed.

Decomposition involves breaking down a problem into smaller problems so they are easier to manage and solve. This can be displayed through structure diagrams, with each individual

task split into simpler tasks. Instead of looking at the problem as a whole, the program can be viewed as:

- Managing accounts and logins
- Gathering inputs
- Scheduling
- Publishing

These can be broken down further (see [1.7](#) and [2.2](#)) to create a meticulously designed solution. Each task is independent of the next, which allows individual modules to be rigorously tested without others having to work. It also allows more important tasks to be completed first, to ensure the client's satisfaction is maximised. Using decomposition is integral to creating a successful project.

Iteration is a method of programming that repeats tasks several times. This means that one loop, or one press of a button, can automate a task to run repeatedly and accurately. This is useful when sending bulk emails, which is a tedious task when completed manually. Using a program to automate this process would not only save the organiser's time, but also reduce the chance of inconsistencies and allow timetables to be released earlier. This could mean that any issues are spotted while they can still be modified.

Students can be quite forgetful and need reminders for their time slots. On the other hand, some students can get quite stressed and want reminders to confirm it, for some peace of mind. One benefit of computers is their precision in timing, unlike humans who may send the reminder email late or forget to send it altogether. A reminder email sent to a large number of people at a certain time makes use of this capability, as emails can easily be sent at an exact time with minimal effort.

## 1.3 Stakeholders

My client is Ms Pearcey, who organises the school's music examination entries and music lesson software. However, future clients may include music teachers across the country who organise music exams. My stakeholders are:

- Exam organiser - they will be the primary users of this software, using it to create timetables quickly and easily. This will make their work more efficient and accurate, and will allow all other stakeholders to receive vital information earlier.
- Students - the students who take the music exams will be affected by this software, as they will reliably receive email reminders and be able to log in to the system to check the time of their exam. This will simplify the process and allow them to focus on their exam.
- Accompanists - the role of accompanists is to play with the students during their exam (if needed), so they must be punctual and prepared for the occasion, just like the students. Therefore, they will also be able to log in and view the exams which they are accompanying, as well as being notified by email.
- Music teachers - as well as reminding students about their exam timeslot, music teachers must prepare students for the exam. Therefore, they will benefit from the quicker scheduling process, as it allows them to be involved in the student's upcoming exam for a longer period of time.
- Examiners - It is important to note that examiners are external, and vary each exam day, so it is inefficient to provide the examiner with a login. The exam organiser usually liaises with the examiner by phone and shares details, such as the timetable, beforehand. This communication can be streamlined due to faster and more accurate scheduling, and the examiner will also indirectly benefit from less missed exams as a result of improved reminders.

Usually, all school staff are informed on the day of exams during the teachers' morning briefing, to try and keep people quiet around the music rooms. Although it would be more effective to email staff beforehand, it would likely be seen as excessive and even irritating for the members of staff to be told repeatedly.

### 1.3.1 User group

#### *Ms Pearcey (Exam organiser)*

Ms Pearcey is my main client for this project. As someone who has been scheduling exams for over 20 years, her experience will allow her to give accurate feedback to the project's development. She teaches percussion at my school, organises all music exams, teaches music as a subject, and is even an examiner for these music exams. She will be the one using this software the majority of the time, using it for its main function: to schedule music exams. This software will help her to save time while completing this process, and reduce the likelihood of unsuccessful timetabling. Throughout the project, Ms Pearcey will be able to give resources and honest feedback, helping me to continually develop my project to meet her requirements.

### *Cyrus and Darius Maleki-Toosserkani (Students)*

Cyrus and Darius are aspiring professional guitarists, taking music lessons weekly and regularly taking music exams. After years of practice, both have achieved grade 7 in guitar and are working towards grade 8. They have experience being on the other side of this project, so they will provide a fresh perspective and thorough insight into what students would like to see implemented in this project. Cyrus also has an extra-curricular interest in graphic design, and has agreed to help with the visual aspects of the project such as logo design.

### 1.3.2 Stakeholder interviews

#### *Ms Pearcey*

I was able to conduct a comprehensive 45-minute interview with Ms Pearcey during the early stages of my project, where I developed a good understanding of the problem, stakeholders, requirements and possible limitations. The full interview transcript can be found in the [Appendix](#), but I have condensed the information here to emphasise the main takeaways from the interview:

- Understood the current process for scheduling a music exam, outlined in figure 1.1.1a
- Accompanists are internal (usually music teachers), whereas examiners are external (from the exam board). Therefore, accompanists can receive a login to the proposed system, but examiners cannot.
- When schedules are made, Ms Pearcey tries to group each accompanist's exams together, but after some time the accompanist needs a break, so a different exam will be added to allow the accompanist to rest. The scheduled breaks normally act as a suitable break for the accompanist as well, so this need not be included in the scheduling algorithm.
- There are 2 exam boards used by the school: TCL (Trinity College London) and ABRSM. These exam boards differ in many ways, which must be taken into account. However, Ms Pearcey later informed me that our school uses ABRSM less than TCL, and would rather see TCL exam scheduling working well than both working at a substandard level. She agreed that each exam day and therefore each timetable is for one exam board. Each exam board differs in:
  - Length of exams for each grade
  - Length and timing of breaks
  - Application process and the online portal used to apply
  - Deadlines set
- For both exam boards, the online portal does not let you import anything (e.g. databases). Therefore, the created timetable must be manually inputted into the system by the exam organiser.
- The produced schedule is converted into a PDF file by Ms Pearcey. She then puts this printout on the notice board, and then manually emails students, their parents and music teachers, as well as accompanists. She mentioned it would be 'ideal' to have an online system that everyone can log in to and that sends out emails automatically. This would speed up the process of sharing the schedule, allowing all stakeholders to be informed earlier and giving them more time to identify problems.

- She said that people have missed and been late to exams in the past, as they have forgotten the time. This highlights the need for better reminders, through automatic emailing.
- Each examinee may or may not need to have an accompanist. If they do need an accompanist, they may require a specific accompanist, or the general 'school' accompanist. The school accompanist is one person who accompanies most exams. Each exam cycle may require a different school accompanist. Therefore, each examinee will either have no accompanist, a specific accompanist, or the 'school' accompanist.

After further discussions with Ms Pearcey, I have clarified this process further in [1.1.3 Updated TCL regulations](#).

### *Cyrus and Darius Maleki-Tooserkani*

To gain a better understanding of the problems faced by students and their requirements for this project, I made a short survey for Cyrus and Darius to complete. Question 1 establishes proficiency in their musical knowledge, which helps me to understand the reliability of feedback when evaluating the success of the project. The remaining questions allow me to understand their experiences with exam scheduling and gather their opinion. Question 4 was asked following Ms Pearcey's interview, where she informed me that, for the TCL exam board, she must collect all entries 2-3 months before the date.

#### **Questions for Cyrus and Darius**

1. What is the highest grade you have achieved?
2. How do you normally find out about exams?
3. Would you like to be asked for availability?
4. If asked for availability 2-3 months in the future, would you want every student to be asked for availability?
5. Do you always make it to exams 15 minutes before (as instructed)?
6. Would you want a reminder email for your exam?
7. What is your opinion on the whole music exam process and how could the process of scheduling exams be improved?

#### **Response from Cyrus**

1. Grade 7 in classical guitar
2. I am usually told by my music teacher.
3. No, as it's highly unlikely the exam will happen at the same time of something with similar importance
4. No, it would get too difficult to coordinate that far into the future, especially for younger students who are less organised. It would be simpler to not ask for availability, and just give students a timeslot.
5. Yes
6. No, because the exam's importance means that it is difficult to forget.
7. It's efficient, there's nothing in particular that could be improved

Cyrus did not want to be asked for availability, as the exam would usually be more important. Interestingly, he later raised the point that students, especially younger students, are not

organised enough to know their availability 2-3 months in advance, which is when this information would be collected. Ms Pearcey also raised this point as the reason for why student availability is not collected before making schedules.

### **Response from Darius**

1. Grade 7 merit in classical guitar
2. Through my music teacher.
3. I don't mind as the exam doesn't really affect anything except missing a lesson, however I once had to move a dentist appointment which was at the same time as an exam which was frustrating.
4. No, as things could change in 2 or 3 months so I think a month before would be better. If there was a school trip or appointment in 3 months it is possible that I would forget to consider that.
5. Yes
6. Yes, it would confirm the time for me and reduce some stress of the exam.
7. I think the process is good in terms of informing, although the email reminder sounds like a good idea, and scheduling. To improve, maybe a specific day could be chosen with higher priority given to people taking higher grades.

Not being asked for availability led to 'frustrating' consequences for Darius, but he also thought that 2-3 months is too far into the future to ask, saying he could forget to mention a school trip or appointment that clashes with the exam. He later mentioned a possible improvement that higher grades would have more priority with their choice of exam time. While this method could improve scheduling for the higher grades, it may lead to the accompanist's time being wasted, but nevertheless an interesting idea. Darius also believed that an email reminder would be a good idea, saying it would 'reduce some stress', which can lead to better exam performance. It is important that a solution is developed to remove the barriers that students like Darius face, because it provides a clear path to a successful musical career, allowing them to flourish.

### ***Recent examinee survey***

I wanted to gather a broader perspective of students that have recently taken music exams, over a range of years and instruments, so I created a short Google form that was sent to recent examinees about their experience. I received 20 varied responses, which confirms the validity of these results, giving me more confidence that the requirements of students are heard. All questions and responses are in the [Appendix](#), in the same format as shown in Figure 1.3.2a

Responses: Music Exam Student Survey													
Q4													
1	Have you recently completed a music exam?	Which grade was this exam for?	When applying for the exam, were you asked for availability?	Did you have any issues with the time slot you were given? (ie. unavailable at that time)	If yes: Please explain the issue and how you fixed it.	Were you on time to the exam?	How did you find out about your exam time slot?	What is your opinion on how you were told about your exam time slot?	How could the process of taking music exams affect your lessons?	Have you ever missed a music lesson?	How has Practice Pal affected your lessons?		
2													
3	Yes	5	No, but I hope they did ask for availability	Yes, but I did nothing about it	I had dental surgery two weeks ago	Yes, I was there 15 min to Music notice board	Very happy			Ocassionally (once or twice)	I dont know what that is		
4	Yes	3	No, but I hope they did ask for availability	No		Yes, I was there 15 min to Email to me, Email to my Very happy				Ocassionally (once or twice)	I dont know what that is		
5	Yes	2	No	No		Yes, I was there 15 min to Email to me, Music teach Very happy				Ocassionally (once or twice)	I dont know what that is		
6	Yes	1	Yes	No		Yes, I was there 15 min to Music notice board	I would have liked an em			Lots of times, I always to	I use the Practice Pal app		
7	Yes	5	No, but I hope they did ask for availability	No		Yes, I was there 15 min to Music notice board	I would have liked an em			Ocassionally (once or twice)	I dont know what that is		
8	Yes	1	Yes	No		Yes, I was there 15 min to me, Music notice Very happy		An email reminder the day before		I dont know what that is			
9	Yes	7	No	No		Yes, I was there 15 min to Music notice board, Musi Very happy, I dont check An email to both parent a	Ocassionally (once or twice)			Ocassionally (once or twice)	I dont know what that is		
10	Yes	4	No, but I hope they did ask for availability	No		Yes, I was there 15 min to Email to me, Music notice Very happy				Ocassionally (once or twice)	I dont know what that is		
11	Yes	5	No	No		Yes, I was there 15 min to Music notice board	I would have liked an em			Lots of times, I always to	I dont know what that is		
12	Yes	5	No	No		Yes, I was there 15 min to Music notice board, My t Very happy				A few times	It is useful to get an em		
13	Yes	5	No, but I hope they did ask for availability	No		Yes, I was there 15 min to Music notice board	I would have liked an em I would appreciate if they			Ocassionally (once or twice)	I dont know what that is		
14	Yes	5	No	No		Yes, I was there 15 min to Music notice board	I would have liked an em			Ocassionally (once or twice)	I dont know what that is		
15	Yes	7	No, but I hope they did ask for availability	No		Yes, I was there 15 min to Email to me, Music notice Very happy				Ocassionally (once or twice)	I dont know what that is		
16	Yes	5	No	No		Yes, I was there 15 min to Music notice board	Very happy	Emails sent with exam fir	Ocassionally (once or twice)	I dont know what that is			
17	Yes	4	No	No		Not necessarily I wasn't Yes, I was there 15 min to Music notice board	I would have liked an em: First come first serve slot A few times			I like getting emails for ex			
18	Yes	6	No	No		Yes, I was there 15 min to Music notice board	Very happy			Ocassionally (once or twice)	I dont know what that is		
19	Yes	3	No	Yes, but I did nothing about it	it was on a strike day and Yes, I was there 15 min to Music notice board		I dont check my emails o having you let a second	Ocassionally (once or twice)	I prefer having it on the n				
20	Yes	3	No	No		Yes, I was there 15 min to Email to me, Music notice Very happy		Perhaps in future student	Ocassionally (once or twice)	I dont know what that is			
21	Yes	3	No	No		Yes, I was there 15 min to Email to me, Music notice They got the times incor			A few times	I dont know what that is			
22	Yes	3	No	No		Yes, I was there 15 min to Email to me, Email to my Very happy	N/A			Ocassionally (once or twice)	I like getting emails for e		

Figure 1.3.2a : Spreadsheet created from responses to the survey (see Appendix for full spreadsheet)

### Summary

32% of examinees hoped that they were asked for availability, and 10% of examinees had an issue with their time slot but did not act on it. This involved ‘shortened recovery time’ from surgery, which decreased this student’s abilities in the exam. This clearly shows that not collecting student availability has caused detriment to a student’s exam performance, so the proposed solution should make an effort to change this. Even if availability may not be collected in the final solution, the student should feel encouraged to inform the organiser of any concerns.

45% of students said that they would have liked an email reminder sent to either them or their parents, just as Darius said in his survey. Some of the respondents emphasised this point later in the form, saying it would improve the exam process. It became apparent that an email reminder was one thing missing that many students would appreciate more. Among this 45%, the majority were in younger years, suggesting the younger students are less effective at keeping organised, a point Cyrus raised in his interview. An email one day before the exam would be appropriate to the needs of these students, preventing them from forgetting this event and saving the time of all involved.

Only 1 student claimed they have never missed a lesson, with 70% missing 1 to 2 per term, and 10% ‘always forgetting’. This emphasises that students can be forgetful, validating the need for an email reminder. Some music teachers have started to use the software Practice Pal, which sends an email reminder the day before a lesson, among many other functions (see [1.4.2 Practice Pal](#)). Out of the students that had heard of Practice Pal, 60% found that the emails have helped them to remember lessons more often. This clearly shows that an email reminder for exams would be effective, as it is already showing positive feedback with lessons. Regular and automatic emails are accurately implemented by a computer, signifying that a computational approach would be best. However, this finding has clearly verified this idea in effect, proving that using a computer is the best solution to this issue.

## 1.4 Research of existing solutions

### 1.4.1 Exam board portals

After scheduling the exams, the organiser must input and submit this information into the exam board's online portal. The system forces exam timetables to conform to their regulations by adding each student individually. Unfortunately, there is no other way to input data (ie. databases) so it is not possible for my program to communicate with it. Therefore, the best method is for my system to initially create a schedule that meets the exam board's standards, to avoid further complications.

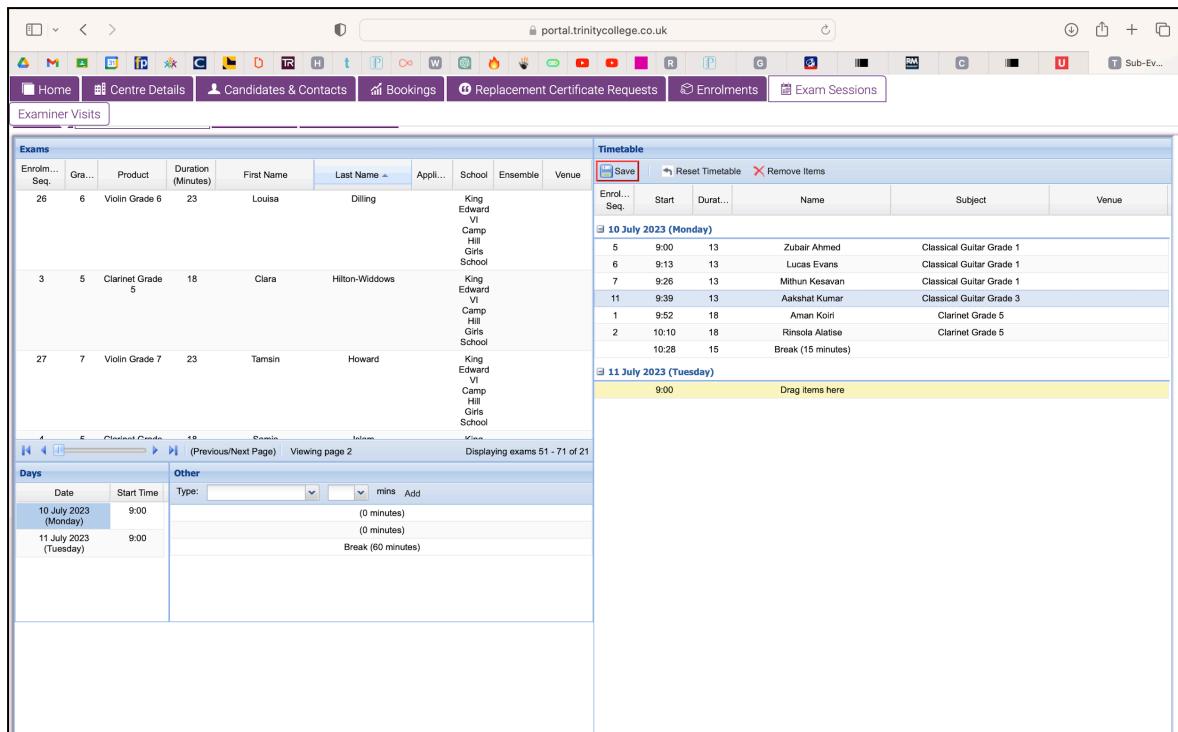


Figure 1.4.1a: TCL portal for submitting exam schedule, provided by Ms Pearcey

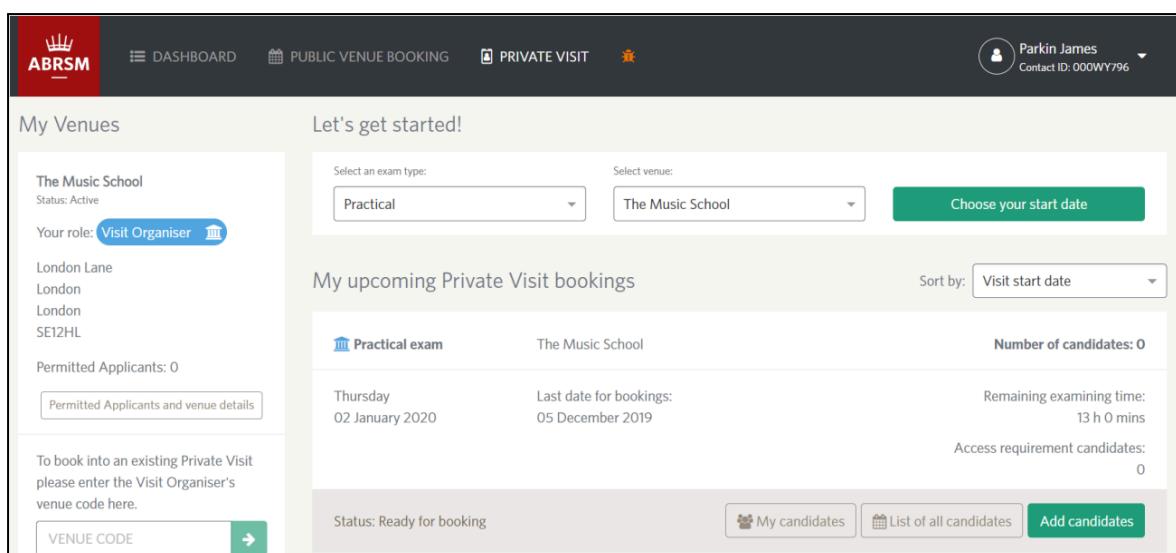


Figure 1.4.1b: ABRSM portal for submitting a schedule, provided by the exam board.

These exam boards provide this portal to force schedules to conform to their rules, with minimal help to schedule exams. They simply exist as a means of gathering information, rather than aiding in the difficult process of scheduling exams. Furthermore, they are extremely rigid, with no method of importing pre-made schedules. While the exam organiser will always have to use this system, an additional program would help simplify this process and avoid complications.

Both portals allow students to be added individually, which I believe unnecessarily wastes time. A quicker method would be to create a spreadsheet of entries with appropriate details (name, grade, accompanist etc.) and import that into the system. Ms Pearcey mentioned that she collects entries through Google Forms which automatically creates a spreadsheet of responses, therefore it would be simpler to use this method. A significant drawback with this portal is that it forces the user to timetable exams themselves, by dragging and dropping exams across the day's calendar. Therefore, automatic timetabling is the key feature my proposed system should implement, using Ms Pearcey's criteria to create an efficient schedule. However, it is still important to give the exam organiser the opportunity to manually adjust this timetable, as these portals have done.

Once schedules have been saved, the exam organiser can still access the timetable until the day has passed. I liked this functionality, however I would want to extend this to students, accompanists, and music teachers. I believe it is critical to give all relevant people this information as quickly as possible, so if the schedule is changed, it would automatically be changed on the system for all to see.

Ms Pearcey mentioned that the exam board portals do not create a basic PDF of the schedule, forcing her to manually create this. From the recent examinee survey I conducted, 95% of students found out their time slot through this printout on a notice board, proving that this feature is vital for students. Therefore, it seems sensible for my program to automatically create a PDF of the timetable, preventing Ms Pearcey from creating it herself. This feature is supported by libraries, allowing me to develop code to utilise these libraries and create a PDF easily. As the file always has the same format, and is automated by libraries, it is definitely suitable to include in my proposed solution.

It is important to note that Ms Pearcey mentioned that the TCL exam board is much more popular than ABRSM for students in our school, and is widely used in schools across the nation. Due to the inflexible and meticulous nature of exam board rules, we agreed that it is absolutely essential that these rules are met. Therefore, it would be more sensible and beneficial for my client that I focus this project on just the TCL exam board. By refining a single scheduling algorithm to comply with their set of rules, I am able to generate more efficient and practical timetables than by splitting my workload into 2 potentially mediocre scheduling methods.

#### 1.4.2 Practice Pal

Practice Pal is software designed for scheduling music lessons. It is very complex, with many different features implemented for the ease of the scheduler. While it doesn't provide

exam scheduling capabilities, it is very similar in nature, allowing me to analyse its interface and much of its functionality. The complexity of Practice Pal is beyond the scope and timeframe of this project, so I cannot hope to incorporate all of its features in my solution, only a select few. Ms Pearcey has kindly provided me with screenshots of the system from the perspective of an organiser/ teacher.

The basic interface is a calendar for the week (figure 1.4.2a). I particularly like how simply the system displays its features, focussing on the important aspects as to not overwhelm the user. There is a highlighted button to add a lesson, bringing the user to figure 1.4.2b. It says 'select learner', prompting the user to select the student for whom the lesson is for. This is done through a drop-down list of students, taken from each teacher's personal list.

The screenshot shows the 'Lessons' page of the Practice Pal software. On the left is a dark sidebar with the 'Practice Pal' logo and the name 'Lorne Pearcey'. Below the logo are several menu items: 'Lesson planner', 'Your learners', 'Groups', 'Ensembles', 'Attendance' (with a checked checkbox), 'Payments', and 'School claims'. A 'Lesson room' button is highlighted with a white border. At the bottom of the sidebar are links for 'Dashboard', 'Settings', 'What's new?', 'Help', and 'Logout'. The main area is titled 'Lessons' with a blue 'Add' button. Above the calendar grid are buttons for 'Copy Fr...', 'Schedule week', and a dropdown menu. The calendar grid shows time slots from 08:00 to 13:00 for each day from Monday 17 Jul to Sunday 23 Jul. Each slot contains a small icon. At the bottom right of the grid are 'Export' and 'Sync' buttons for Excel, iCal, and Google.

Figure 1.4.2a: Interface for scheduling lessons with Practice Pal.

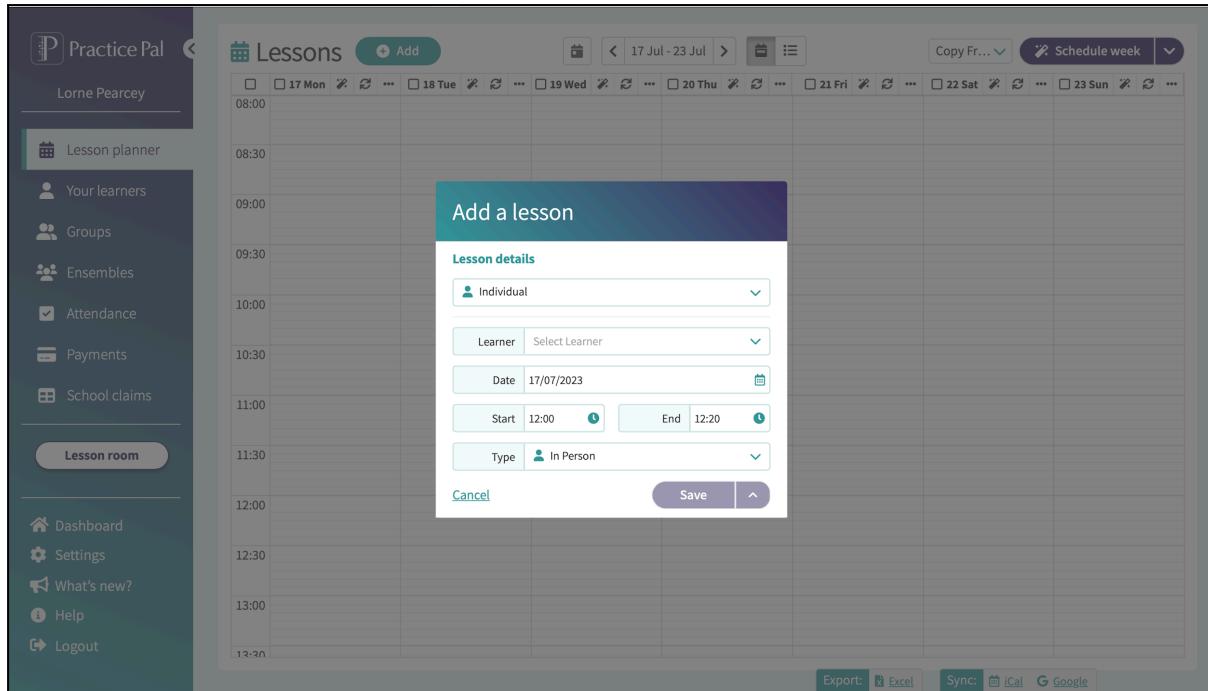


Figure 1.4.2b: Adding a lesson by selecting from a list of students.

Associated with the students is a wide range of information, including their full schedule week by week (figure 1.4.2c). Practice Pal easily integrates with the school management information system (MIS) (figure 1.4.2d). For my school, this software is SIMS. This makes it easy to add students, gather information and inform relevant staff. In the context of exam scheduling, using MIS integration would allow me to easily gather inputs. However, creating the framework for 14 different MIS as Practice Pal has done is extremely difficult within the timeframe of my project. Additionally, the difficulty of collaborating with these companies to reach a solution poses another problem. A possible alternative would be to allow my system to import data from common database formats, such as .csv. This would allow the organiser to export data from any MIS to create the database, then import it into my system. While it is slightly more cumbersome for the user, it allows any MIS to communicate with this system, albeit less directly.

The screenshot shows a weekly timetable for 'Lorne Pearcey'. The sidebar on the left contains links for 'Lesson planner', 'Your learners', 'Groups', 'Ensembles', 'Attendance', 'Payments', 'School claims', 'Lesson room', 'Dashboard', 'Settings', 'What's new?', 'Help', and 'Logout'. The main area displays a grid of lessons from Monday to Friday. A specific lesson for 'Rohan Desai' on Thursday is highlighted with a red box, showing details like time (10:35), subject (12L1), and grade (99E). Other lessons listed include Computing, Further Maths, Physics, and Study.

Figure 1.4.2c: My timetable shown in grey blocks upon clicking my name, this data is pulled straight from SIMS.

## MIS Integration

Find every system we integrate with along with helpful setup guides on the [Wonde website](#)

The page displays logos for various MIS systems: Arbor, isams, Bromcom, WCBS, SIMS, SchoolBase, schoolpod, RM, pupilasset, IRIS, engage, ScholarPack, and VSWARE. Each logo is accompanied by its respective company name in a stylized font.

Figure 1.4.2d All school management information systems (MIS) that integrate directly with Practice Pal, such as SIMS which is used by my school.

Music lessons this week					10 Jul - 16 Jul	Practice Pal	
Monday 10th		Tuesday 11th		Wednesday 12th			
Catherine Butler		Alexander Hay	Danielle Palmer	Chris Hickman	Elizabeth Goble	Adrian Rose	
08:30 Xander Davis, 8K 09:00 Caleb Wilton, L2 09:20 Ben Law, 9C 09:40 Zaid Rassam, 9B 10:00 Henry Bloomer, 7K 10:20 Pascal Fernando, 8H 10:50 Lemuel Adjei, 9B 11:10 Isaac Williams, 7K 11:30 Bashir Said, 8H		08:00 Vanish Pradhan, L1 08:30 Tony Zhou, 8H 08:50 Sameen Jan, 7K 09:10 Zain Piracha, 8E 09:30 Pramath Murthy, 10H 09:50 Sathy Vaideyanathan, 9C	10:00 Muhammad Haider, 8H 10:20 Vidyut Tutika, 10E 11:00 Aashkhat Kumar, 10C 11:20 Yusuf Ali, 7K 12:00 Sikandar Sajid, 9H 12:40 Vanish Pradhan, L1 13:00 Muhammad Haider, 8H 13:40 Nikhil Gillian, 8K 14:20 David Bai, 10C 15:00 Mithun Kesavan, 8K 15:20 Lucas Evans, 8E 15:50 Zubair Ahmed, 8E	10:35 Ben Hodgetts, 10K 10:55 Finn Cambridge-Davies, 10E 11:15 Aditya Manu, 7E 11:35 Samuel Wiseman, 8K 11:55 Cheuk Yin Lam, 8K	10:00 Daniel Okpla, 9K 11:20 Ritvik Gautam-Sanani, 8K 11:40 Manav Sethuraman, 8E 12:00 Ibrahim Miah, 10C 12:20 Hrish Narayanan, 9C 12:40 Sri Grandhi, 9H 13:00 Levi Dandy, 9C 13:20 Adam Bashir, 9E 13:40 Nethiran Mugurtha, 10H 14:00 Aditya Krishna, 10H		
Elizabeth Goble		Erica Newton	Elliot Drew	James Griffith	Linnea Markgren	John Meadows	
09:30 Ben Hone, L8 10:00 Alexander Hopkins, 10K 10:20 Harley Hau, L8 10:40 Yicheng Hu, 9C 11:00 Taheen Islam, 10E 11:20 Isa Naveed, 9K 11:40 Milan Patel, 8E 12:00 Rachit Sehdev, 8C		13:00 Aneeq Naveed, 10K 13:20 Ren Zhi Howse, L5 13:50 Dexter Guest, 7K 14:10 Kevin Cao, 7B 14:35 Omer Arshad, 7E	10:20 Samuel Nohov, L2 - GIRLS S...  10:30 Ethan Ho, 10H 10:50 Lucas Ribeiro, 7C 11:10 Anirudh Arbyamir, 7H 11:50 Elias Griffith-Al, 7H 12:10 Zane Shah, 9H 12:35 Guitar Ensemble 3 2+ 3 learners	10:00 Adrian Ifrim, U2 09:20 Ryheem Miah, 9E 09:40 Finlay Guevara, 10E 10:00 Abbas Rizvi, 7K 10:20 Zihan Oscar Zou, 7K 10:40 Jesse Kunbor, 10C	10:30 Ben Hone, L8 11:00 Alexander Hopkins, 10K 11:20 Taheen Islam, 10E 11:40 Yicheng Hu, 9C 12:00 Harley Hau, L8 12:20 Rachit Sehdev, 8C 12:40 Milan Patel, 8E 13:00 Isa Naveed, 9K	10:00 Harshith Salanke, 10H 10:20 Jed Beeston, 9B 10:40 Jaesh Manivannan, 8E 11:00 Christopher Moore, 8H 11:20 Fred Belcher, 7H 11:40 Jacob Hannatty, 7E 12:00 Aman Tomas, 10E 12:20 Aman Koiri, 10E	
Erica Newton		James Griffith	Linnea Markgren	Lorne Pearcey			
13:00 Macca Ros-Nalugon, 9B 13:25 Benjamin Wrangels, 7C 14:35 Bastian Aberg, 7H 15:05 Alexander Cammiss, 9K		10:10 Ethan Ho, 10H 10:50 Mustafa Ahmed, 9C 11:10 Anirudh Arbyamir, 7H 11:50 Elias Griffith-Al, 7H 12:10 Zane Shah, 9H 12:35 Guitar Ensemble 3 2+ 3 learners	08:40 Aiden Arul, 10E 09:15 Muteerahmaan Shah, 9C 09:35 Lemuel Adjei, 9B 09:55 Mihran Khan, 8C 10:15 Zain Steers, 7C	08:15 Shawn Sen, 7H 12:35 Gus Loevengren Tremlett, 8K 12:55 Adnan Lohawala, 9C 13:15 Muhammad Qamar, 9H 13:35 Mateusz Grzeslik, 7C 14:20 Mihran Khan, 8C	08:15 Jed Jimenez, 10C - Ensemble ... 08:35 Rafael Kenny, 10E - Ensemble...		
Zhivko Georgiev		Linnea Markgren	Lorne Pearcey				
09:30 Yusuf Sultan-Qurayshi, 8K							
<a href="#">More lessons below</a>		<a href="#">More lessons below</a>	<a href="#">More lessons below</a>	<a href="#">More lessons below</a>	<a href="#">More lessons below</a>	<a href="#">More lessons below</a>	

Figure 1.4.2e: Weekly calendar of all lessons in school, easily accessible online.

Once the schedule is created, it is published by clicking a button (see figure 1.4.2c). This function updates the calendar on the server, which allows anyone to immediately view the latest schedule (figure 1.4.2e). I like this functionality because it keeps students up to date, minimising the risk of confusion and giving them more time to raise potential problems. This was highlighted as a major issue with the current method by Ms Pearcey in her interview, saying a feature like this would be ‘ideal’. Furthermore, publishing will automatically send emails to all students, informing them of their lesson (figure 1.4.2f). If a change must be made, it is then re-published, and any students affected by the changes are notified. This makes the system very easy to use with just one button causing mass emails to be sent and schedules to be updated, which I thought was quite intuitive.

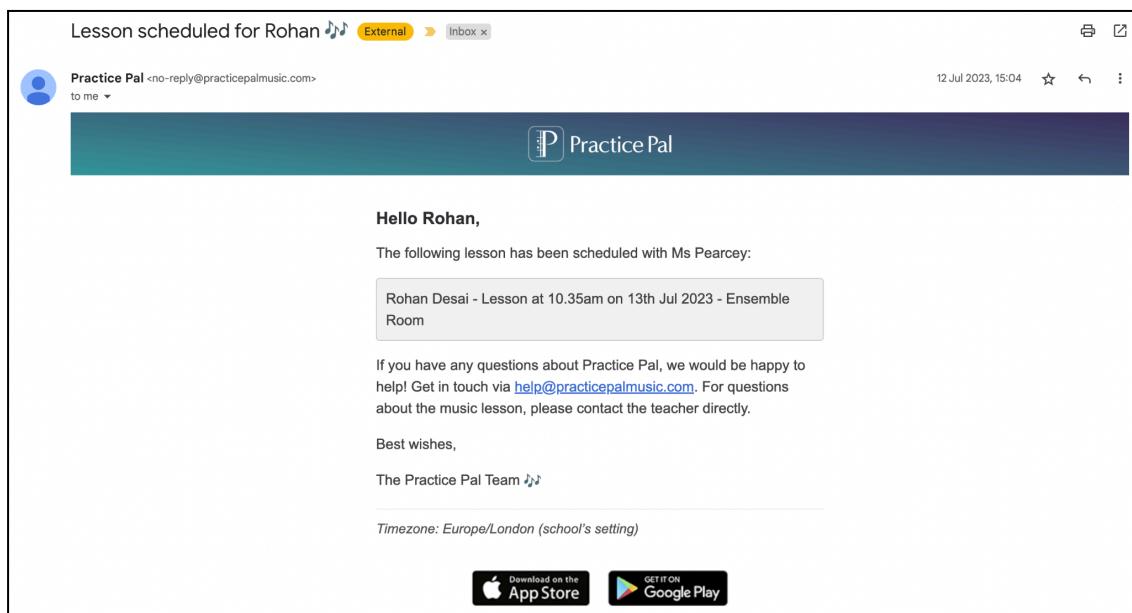


Figure 1.4.2f: Email received by a student informing them about their lesson time.

One interesting function is called 'magic scheduling', which assesses how long each student has been out of each 'normal' lesson previously, and arranges things to balance this out. With a push of a button (figure 1.4.2a), all the lessons are immediately added to the calendar as per the algorithm, then the teacher is left to edit this as she sees fit. Although the algorithm will be different, I would aspire to have a similar level of simplicity with my program's scheduling function. I like how the magic scheduling function is treated as a function rather than a critical aspect of scheduling, as it means that the user still retains full control, with the function being more suggestive rather than forceful.

Year	Subject Restriction
Year 7	Games, PE
Year 8	Games, PE
Year 9	Games, PE
Year 10	Games, PE
Year 11	Games
Year 12	Games
Year 13	Games

Figure 1.4.2g: Music lessons can be marked as off-limits during a certain subject lesson.

Students can be restricted from having a lesson scheduled during certain subject lessons. As shown in figure 1.4.2g, Games and PE lessons detected in a student's timetable on SIMS are off-limits for music lessons. A similar feature would be helpful when scheduling exams as you could mark when an accompanist was not available, except it would be based purely on time. Upon further discussion with Ms Pearcey, she confirmed that this feature would be useful. I found it intriguing that you mark when *not* available. Possibly, asking the user to opt out could increase their availability, making it easier for the algorithm and organiser to schedule lessons.

### 1.4.3 Furlong Maestro

Unlike the previous two solutions, Furlong Maestro is not used by anyone in my user group. This, as well as a lack of online reviews and information makes research difficult. Furlong Maestro is similar to Practice Pal, with many functionalities for managing music lessons. However, it schedules music exams for ABRSM and TCL exam boards, making it much more relevant to this project. I will be focussing on features that differ from Practice Pal, to avoid repetition.

The interface is less aesthetically pleasing as Practice Pal, and seemed to be less intuitive to use. While Practice Pal had a methodical approach that was conveyed by the program, Maestro seemed to clutter the interface with functions being less organised. For example, the key functions in Practice Pal are emphasised with colours (figure 1.4.2a), but with Maestro this is not well executed. The lack of colour coordination forces users to spend more time finding the next feature, making them more frustrated and disoriented. With Practice

Pal, colours are also used to highlight the sidebar, whereas there is no permanent sidebar with Maestro, which confuses the user and makes each new section look foreign. As most users will not be passionate to use computer programs, it is crucial to keep the interface as simple as possible.

I did think that the feature of detecting clashes was quite impressive, it highlighted any issues without forcing the user to do something they didn't want to do. This is a great example of a computational solution that assists the user rather than taking full control. I would like to explore the possibility of doing something similar in my project. However, I did not enjoy the aesthetics of how this clash was displayed. The exclamation mark, as shown in figure 1.4.3a, is not particularly elegant.

The screenshot shows a software interface titled "Editing Read, Jackie (Oboe) - 28/03/2022". On the left, there's a sidebar with various filters and categories. The main area is a table with columns: Pupil, Day, Start, Duration, End, Reg, Room, Discipline, Last Grade, Mark, Classification, and Role. A row for "Applegate, Leon / 12B / LXM / Oboe" has a conflict highlighted with a red border and an exclamation mark icon. A tooltip says "Clashes with: English, 14:00 - 15:30". Another row for "Brady, Pieter / 11E / Oboe" also has a conflict with the same tooltip. At the bottom, there's a "Save Schedule" button and a "Save Current" button.

*Figure 1.4.3a: Scheduling lessons with Maestro. MIS integration detects clashes, which are executed with an inelegant '!'.*

Unfortunately, there was minimal information on the exam scheduling feature, apart from a basic description:

*"Manage and schedule external examinations  
ABRSM and Trinity examination boards  
Communicate timetables to all and manage charges from exam unit base-data"*

This means that I could not analyse their exam scheduling feature, except one interesting aspect of the automatic emailing process (assuming this is what is meant by 'communicate timetables'). The entire email was completely customizable (figure 1.4.3b), with in-built functionality of using variables to inform the recipient of specific information. I liked how much control this gave the user, but I would question the need for this feature, as it introduces another layer of complexity rather than simply pressing a button to send a default email template. I would prefer my solution to find a middle ground, allowing some customised input, but not so much that the user is overwhelmed.

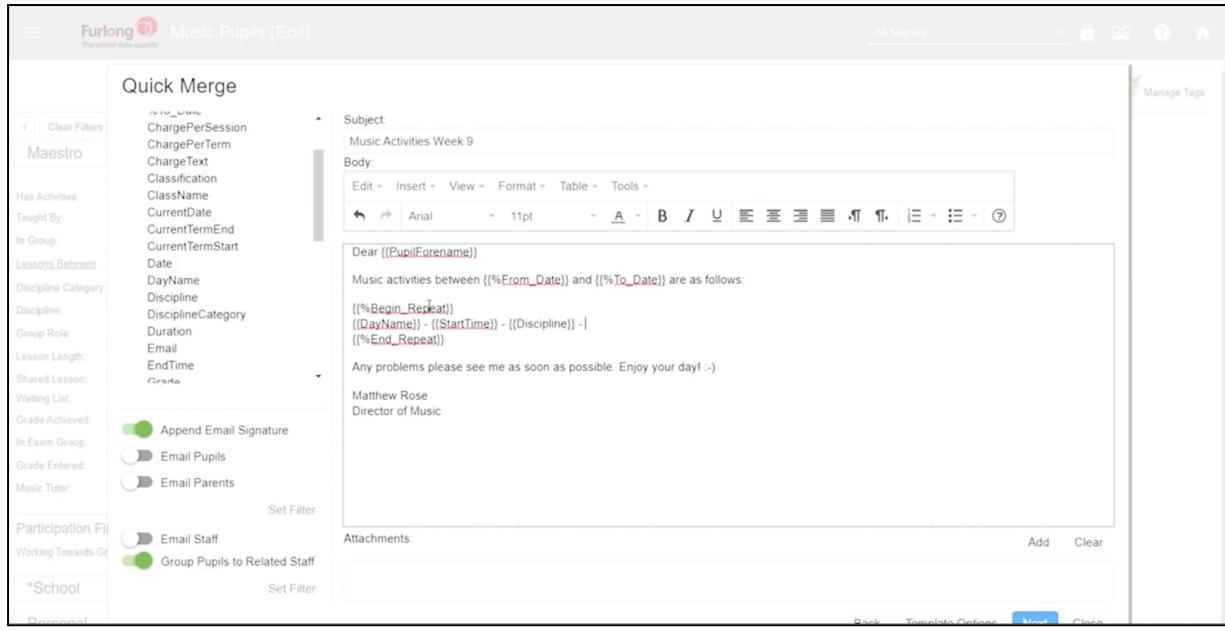


Figure 1.4.3b: Creating an email to send in bulk to all students.

## *1.5 Features of the proposed solution*

Scheduling music examinations involves many processes which can be streamlined and simplified with computational methods. I propose to make a web application which will be used by music exam organisers in schools, as well as students, accompanists, and music teachers. This system will primarily focus on automatically scheduling music exams according to the given data and exam board regulations, then updating relevant people about this schedule. **UPDATE: I have decided to focus this project on my school only, after encountering several difficulties. This has been addressed clearly in [1.5.5 Limitations](#).**

### **1.5.1 A Web-based system**

This software will be web-based, communicating with a web server to keep all users up to date. As this would be costly to execute, I will be using 'WampServer' to simulate this process during development. WampServer is a Windows web development environment, which allows you to create web applications with Apache web server, PHP and a MySQL database. This allows the central server and any clients to be managed on a single device. In practice, the software would have to be adapted slightly but would function in the same way. By being accessible online, users do not have to download any software which saves storage space. Also, it makes the program accessible to anyone, from anywhere with an internet connection, encouraging people to use it due to its easy accessibility. This is important when dealing with people who may be reluctant to use computers, the simplicity of accessing a web page invites them to use it, as opposed to a complicated download which deters and limits users. Another advantage is the security of data held in web servers, which eliminates the worry of backing up data.

### **1.5.2 Gathering inputs and scheduling**

There are 4 different types of users for this proposed solution: the exam organiser, students, accompanists and music teachers. The primary user is the organiser, as they will be creating the schedule. However I, as well as my stakeholders, believe that it is critical for everyone to receive the most up-to-date information as quickly as possible, to prevent any misunderstandings. Therefore, students will be able to log on and view their time slot, accompanists will be able to view the time slots for which they are accompanying, and music teachers will be able to view the time slots for their students. This provides a central point for everyone to view the current timetable, which can reduce any stress or confusion about timings. Using logins also limits the access for each user, as it can allow Ms Pearcey to do certain processes that other users should not be able to.

The key feature will be to create a timetable for exams, without wasting anyone's time or breaking the exam board's protocols. I would want this feature to be as simple as possible, such as a highlighted button in the same style as Practice Pal (figure 1.4.2a). With the help of Ms Pearcey, I have gathered the following information:

- Accompanists should be grouped together as much as possible. Within each accompanist block, exams should be sorted by instrument family, then by instrument, and then by grade (ascending).
- Some students require an accompanist, some require a specific accompanist, and some do not need an accompanist. There must be support for all of these students in the system.
- Exams tend to move upwards in grade order, but there is no strict need for this
- The exam board has a detailed list of rules for exam scheduling. This has been clarified in [1.1 Problem Analysis](#). This system will support the scheduling of exams from the TCL exam board. ABRSM exam scheduling will not be supported, to allow me to focus my attention on developing a more efficient system for TCL, as this is much more useful to Ms Pearcey.

Based on this information, I will create an algorithm to optimise this process. This will require details of all students that apply to take an exam, as well as accompanist availability. To get accompanist availability, I plan to create a feature for accompanists to submit availability to their organiser, by deselecting 30-minute slots during which they are unavailable. I hope that deselecting rather than selecting will stimulate accompanists to increase their availability, making the role of scheduling easier on the organiser.

To gather student information for scheduling exams, I will allow a spreadsheet of this information to be imported into the system, in a common format such as csv. Any spreadsheet software such as Microsoft Excel or Google Sheets will be able to export the file as a csv, which makes this method flexible with many different systems. Even school management information systems (MIS) will be able to export student information in this way, which reduces the need of integrating with these systems and makes this software accessible for everyone.

The schedule must adhere to the regulations, and be able to distinguish between what is a requirement (following the TCL regulations) and what is optional, but still desirable (grouping accompanists). This allows the program to know when to return an error, and when to keep scheduling. It should try to waste as little time as possible, and schedule as many exams as possible. This is especially important, as the benefit of a computational solution is that an algorithm can iterate over a large number of possible options and find the best one in a very short time. Therefore, there should be sufficient complexity in the algorithm so as many examinees can be fit into the schedule as the constraints allow.

Once the schedule has been created, I would still want the functionality of manually editing this schedule, because the organiser may still have specific requirements that they want to manually implement to their liking. In this case, it would be sensible for the program to detect clashes, similar to Furlong Maestro (figure 1.4.3a). This allows the user to edit the order of exams however they want, but still be informed immediately and accurately if an accompanist is unavailable at a certain time. It should also adjust any breaks automatically, as it is required for the schedule to meet the TCL requirements.

### 1.5.3 Publishing the schedule

Once the schedule has been confirmed (after being created by the algorithm, and after manual edits), I would like a feature to 'publish' the schedule, trying to emulate the simplicity of Practice Pal's methods in figure 1.4.2c. This feature would save the schedule to the server, as well as emailing all relevant information to students, parents, accompanists and music teachers. I would like to incorporate a level of customizability in the emailing aspect. However, I want to maintain the simplicity of a single 'publish' button that automates all tasks at once. My research highlighted the effectiveness of a simple button, rather than multiple pages of inputs. A clear, highlighted function reassures the user, and greatly simplifies their workload. Therefore, once clicking 'publish', I will give the option for the organiser to add text to the email. This text will be displayed on every email. This customizability allows the organiser to convey any messages, while not overwhelming them with an excessive range of options. The 'publish' function would also create a PDF of the schedule, containing relevant information in a similar format as shown in figure 1.5.3a. This PDF can then be printed and displayed on notice boards, which is how 95% of recent examinees from my survey were informed of their time slot. Extracting data from databases can easily allow this printout to be created with a program, due to its regular structure. Another important feature is an email reminder the day before an exam, which can be sent automatically by scheduling these emails during the publish function. This is especially useful for students who may be forgetful, and need that reminder to prevent missed exams.

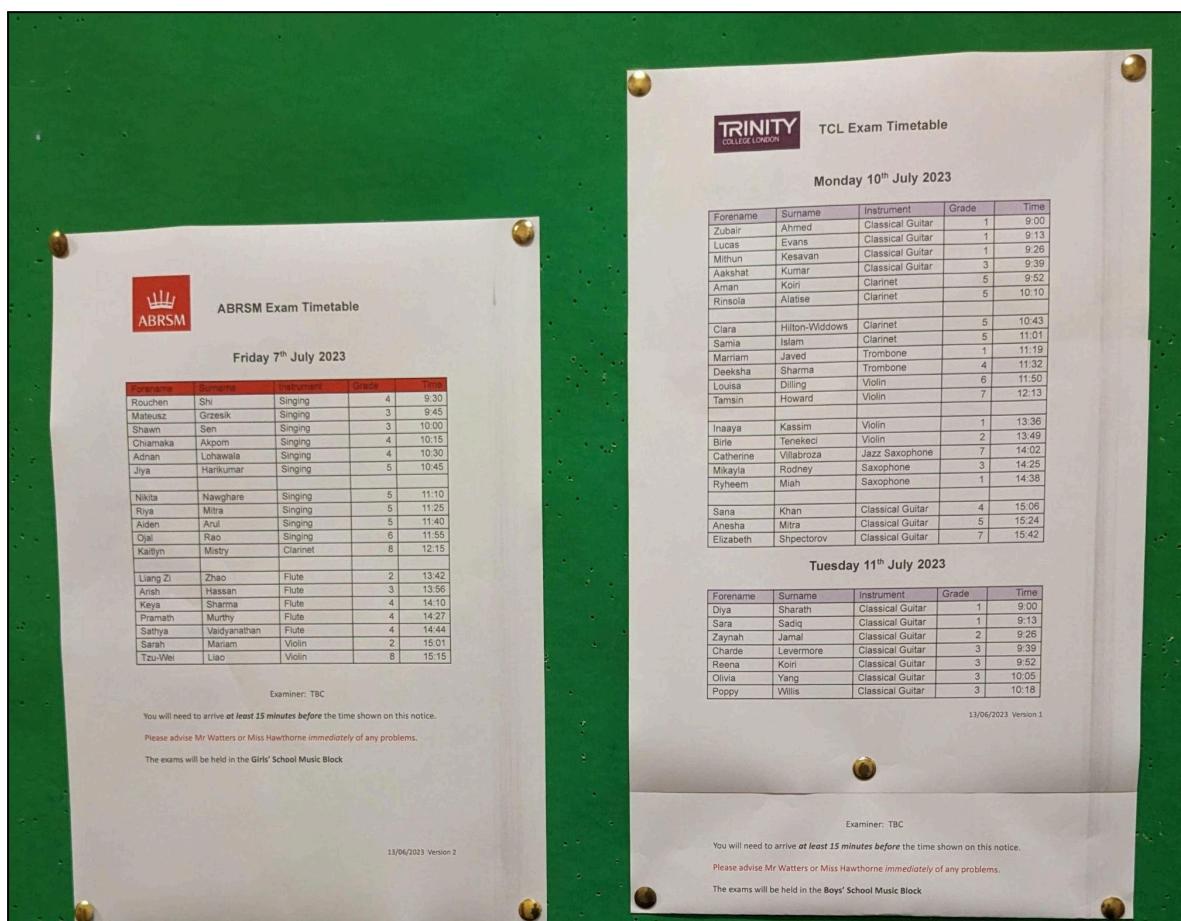


Figure 1.5.3a: My school's music notice board, displaying the schedule for upcoming exams.

#### 1.5.4 Graphical User Interface (GUI)

The GUI must be easy to use and intuitive, as a complex interface can deter users from accessing the useful capabilities of the program. Most music staff will not be particularly interested in using computer programs, and hearing Ms Pearcey's struggles of getting staff to use Practice Pal in our school reinforced the idea that the GUI has to be simple. This can be accomplished by using a colour scheme to make the user more comfortable. Also, using abstraction by hiding any complexity from the user's immediate view can allow their attention to be focussed on the important features. This effect can be emphasised by highlighting the more important buttons to click, and using a permanent sidebar enables the user to keep acclimatised to the system. It is critical that while the interface is simple, complexity cannot be removed, just hidden.

#### 1.5.5 Limitations

Surveying stakeholders revealed that 32% of students would like to be asked for availability, with 10% of students being left unhappy with their time slot. However, I have made the decision that only accompanists will be asked for availability. This may leave students in a difficult situation, as time slots will be given to them, and they will be left anxiously waiting for this to happen. If there is a clash, students will have the chance to voice their concerns to the exam organiser, but my survey showed that no students did this, and instead said nothing. This suggests that students may not be confident to do so, and will remain uncomfortable, which can negatively affect their exam performance. The reason for not including this follows from discussions with my client, Ms Pearcey. She previously considered doing this herself, but decided against it, due to the time constraints and maturity of students. It is very possible that students may act irresponsibly, saying they are unavailable after school and only available during the lessons they detest, for example. The consequence of this would make the scheduling process even more difficult, and could lead to accompanist's time being wasted. Therefore accompanists, but not students, will be asked for availability. I plan to slightly alleviate this problem by making a clear comment on all communications with students, encouraging them to talk to their organiser if they have any problems. As the overall process allows students to be informed quicker, they could potentially have more time to fix any clashes with the exam.

Another key missing feature is MIS integration, which allows my program to directly communicate with the school's database. This would simplify the process of collecting entries, easing their workload even further, and open up the possibility of more helpful features such as emailing subject teachers that their student has an exam during their lesson. However, since there are many different systems used, I would only be able to integrate with a few MISs. This would lead to the program being entirely unusable for some schools therefore I prefer using the common format of csv files. This may prove cumbersome but makes the program flexible for all MISs. Additionally, integrating these systems may become very time consuming, and would raise concerns with data that I can and cannot access during development and testing. I believe this justifies the slight frustration that organisers will experience when submitting student information.

There are 4 different users of this system: the organisers, students, accompanists and music teachers, who will all receive different information once logged in. However, it is common for

music teachers to accompany students, so one person may fall into two categories. This means that one person may have multiple logins, each representing their different responsibilities in exams. While this may be confusing, it allows this person to view all relevant information without ambiguity in their role. Unfortunately, this limitation means that this person will receive multiple emails and have to keep track of multiple login details. This also means that each account will have to be linked to a different email, which is quite frustrating for the user. I would ideally add the option for teachers to also be accompanists within the same account. This would have 2 separate dashboards, with a highlighted button letting users to switch to 'Accompanist mode' or 'Teacher mode'. Unfortunately this would become too complicated with the time constraints of the project, and as this situation is not extremely common, this feature has been removed to allow me to focus on the primary features.

**UPDATE: This problem has been solved during design (see section [2.4 Data](#))**

Due to time constraints, I have decided to omit some less used features. These would be helpful but not to a large extent, therefore I have removed them to make this allow this project to be completed within the time. These features include:

- Managing room allocation, which could let 2 exams to happen at once
- Scheduling exams for SEND students who have longer exam times
- Scheduling with alternative timetables that differ from the standard regulations
- Scheduling Jazz and Music theory exams (not currently available in my school)

These features would require considerable changes to the current system, especially the variance of exam times and regulations. This could distract my attention from the key features of the program due to the difficulty of developing these features, so they will not be included in this solution. However, they will occasionally prevent exams from being scheduled in the ideal arrangement, leaving users dissatisfied.

Another problem with this method is the lack of support with payment. This involves students paying for exams through Ms Pearcey, and students directly paying accompanists. This feature would provide one location for the entire exam scheduling process, making this process as simple as possible. On the other hand, payment processing involves sensitive data, which can raise serious legal and security concerns. I do not want to risk personal information being mismanaged, so there will be no payment functionality in my proposed solution.

**UPDATE:** I originally planned to make this program available for all schools across the country to use. However, upon decomposition of this aspect of the system I have found some difficulties in implementing this which are outside the scope of the project. It would be difficult to securely register a new school, this may involve communications with the school and require more contributions from the exam organiser to keep school details up to date. As one of my key aims is to reduce Ms Pearcey's workload, I must avoid giving her additional responsibilities. The purpose of this project is to solve the problem for Ms Pearcey, and I do not want these issues to distract me from doing so. Therefore, this system will only be available for my school. I later hope to reach out to other schools, so I would still like to make the system suitable for future development.

It would be intuitive for this platform to also communicate exam results to students, however this would distract from the sole purpose of the project, to schedule these exams. As a result, this feature is not included in the software, which forces students to find out through notice boards, or forces teachers to email students individually, which is a laborious process.

***UPDATE:*** *To avoid any confusion, I plan to include a message to students upon login, if nothing is scheduled. The message will read 'If you have recently taken an exam, results will be displayed on the music notice board, or you will be contacted by your music teacher.' This feature is added as success criteria L10.*

## *1.6 Hardware and software requirements*

This system has minimal requirements for users, making it as easy to use as possible.

R	Requirement	Reason
1	A computer with an internet connection	The computer has to access the program through the internet, so a stable connection is needed.
2	Email account	To automate sending bulk emails, the organiser must have an email address. For other users, this is necessary to receive email updates.

## 1.7 Success criteria

Success criteria in this document have been referred to as [letter][number] as shown below, eg. G1 or L6.

### 1.7.1 (G) GUI

G	Criteria	Justification	How evidence will be shown
1	The interface must be simple and intuitive to navigate.	Helps users to use features easily	Screenshots of interface and stakeholder feedback

### 1.7.2 (L) Login

L	Criteria	Justification	How evidence will be shown
1	The product has a login page	Allows users to access their specific information	Screenshots of login system and database
2	A user can only access data they are allowed to access	Prevents users from tampering with the scheduling process	Screenshots of dashboard
3	Users can register themselves	Allows essential data to be stored in the server	Screenshots of registration page
4	The registration page validates information before entering into the database	Prevents SQL injection and ensures sensible data is inputted	Screenshots of registration page
5	Exam organisers will be able to view, create, and delete accounts, with password set upon first login	Reduces risk of inconsistent data by filtering all information through the organiser	Screenshots of 'accounts' page.
6	Email is sent to user upon account creation	To inform the user	Screenshots of email
7	When logged in as a student, accompanist or music teacher, the time slot(s) they are involved in are displayed.	This keeps the information tailored to each person, allowing them to focus on their role.	Screenshots of the main page from a student, accompanist and music teacher login.
8	When logged in as a student, accompanist or music teacher, and nothing	This reassures any users that may be anxious to receive	Screenshots of the main page from a student, accompanist and music teacher login.

	has been scheduled, a message is displayed to say that no exams have been scheduled yet.	their time slots.	
9	When logged in as a student, accompanist or music teacher, there is a short message with the organiser's email included, to email them if they have any concerns.	This reassures the user, encouraging them to voice their concerns (which was a problem in the recent exams, see <a href="#">1.3.2 Recent examinee survey</a> )	Screenshots of the main page from a student, accompanist and music teacher login.
10	For students, if nothing is scheduled yet, another message is displayed : "If you have recently taken an exam, results will be displayed on the music notice board, or you will be contacted by your music teacher."	To prevent students thinking results are shared on this platform.	Screenshot of the main page from a student's login, when nothing is scheduled.

### 1.7.3 (E) Load existing schedule

E	Criteria	Justification	How evidence will be shown
1	Ability to load an existing schedule.	If updates need to be made, the existing schedule must be available to edit.	Screenshot of the main page
2	Existing schedules are deleted at 22:00 on the (final) day of exam.	Once completed, existing schedules serve no purpose. They should be deleted to make space for the next schedule.	Screenshot of the main page before and after 22:00 on this day.
3	There is the option to re-edit and re-publish a schedule, with the extra feature of only emailing those affected.	Prevents duplicate emails which would cause confusion	Screenshots of emails received and 'publish' function

### 1.7.4 (I) Gathering inputs

I	Criteria	Justification	How evidence will be
---	----------	---------------	----------------------

			<b>shown</b>
1	For a new schedule, option to enter basic details such as start date.	This allows the system to gain basic but essential information to inform the scheduling process.	Screenshots of the main page.
2	Able to import and process a csv file of specified student information	Allows all details to be gathered for making schedule	Screenshots of this function and spreadsheet
3	Validation to check if student information is formatted correctly before data is saved to the server.	This keeps the information safe in the server for future use	Screenshots of the main page and the files stored in the server.
4	Accompanist availability can be added manually by organiser	If the accompanist has already informed the organiser about availability, they can be added manually.	Screenshots of availability page
5	Accompanists can be selected from a list and availability is requested	This gives control to accompanists to fill out this information.	Screenshots of availability page, showing all accompanists as a list
6	Requesting availability sends accompanists an email	This keeps them informed about the situation.	Email received by accompanist
7	Status can be seen while waiting for availability to be entered	This allows the organiser to monitor the process easily	Screenshots of the status information in the availability page
8	Accompanists can log on and submit availability as 30-minute slots throughout the day.	This allows the exam to be adjusted to fit their calendar.	Screenshots from the accompanist's login, showing the day calendar as 30-minute blocks.

### 1.7.5 (S) Scheduling

<b>S</b>	<b>Criteria</b>	<b>Justification</b>	<b>How evidence will be shown</b>
1	Creates a schedule that adheres to the TCL exam board regulations	The schedule must stick to exam board rules to be accepted	Screenshots of schedule and testing this against the regulations
2	Schedule ensures minimal time is wasted.	To improve efficiency	Screenshots of schedule and stakeholder feedback
3	As many exams are scheduled as possible	To give as many examinees the	Screenshots of schedule and stakeholder feedback

		opportunity to play	
4	Schedule groups together each accompanist's exams as much as possible	Avoids wasting the accompanist's time	Screenshots of schedules with different inputs
5	Within each accompanist block, the schedule sorts exams by instrument family, then instrument, then grade.	Ms Pearcey and TCL have said that this would be preferred.	Screenshots of schedule
6	Schedule adheres to accompanist availability	Ms Pearcey has specified this as a requirement	Screenshots of schedule compared to accompanist availability
7	Schedule can determine if an accompanist has submitted enough availability	If someone is not available enough, Ms Pearcey will know who to ask for more availability.	Screenshots of errors displayed if someone is too unavailable
8	Schedule is stored securely in server	This allows it to then be accessed by other users	Screenshots of file in the server
9	Opportunity to edit schedule before saving	To give the user control over the schedule	Screenshots of schedule and the edit feature
10	When editing, clashes are detected automatically	This helps user with the scheduling process	Screenshots of editing functionality
11	When editing, breaks are added automatically	Allows the schedule to always conform to regulations	Screenshots of editing functionality

### 1.7.6 (P) Publishing

P	Criteria	Justification	How evidence will be shown
1	The publish function sends mass emails to students, parents, accompanists, and music teachers.	This automates the process of informing others, simplifying the task.	Screenshots of the publish function and emails received from students, parents, accompanists, and music teachers.
2	Emails send specific information with the option for the organiser to add some text.	This allows the organiser to convey any extra messages, and adds	Screenshots of customization feature, and emails received from students, parents, accompanists, and music

		customizability.	teachers.
3	At 5pm the day before the exam start date, a reminder email is sent to students, parents, accompanists, and music teachers	This reminds everyone about their time slot, which can reduce some stress about the exam.	Screenshots of the publish function and emails received from students, parents, accompanists, and music teachers.
4	The publish function automatically creates a PDF of the schedule	PDF files can be distributed easily to anyone who wants access to the timetable	Screenshots of publish function and created PDF.
5	There is an option to download and print the PDF	This allows the PDF to be easily printed so it can quickly be put onto a notice board for everyone to see their time slot.	Screenshots of publish function, created PDF and printing process.

# 2 Design

## 2.1 Problem decomposition

The task of scheduling music exams is quite complex and to help me achieve all of the success criteria I have decomposed the problem. This allows me to tackle each sub-problem individually, which allows me to come to a solution much more efficiently.

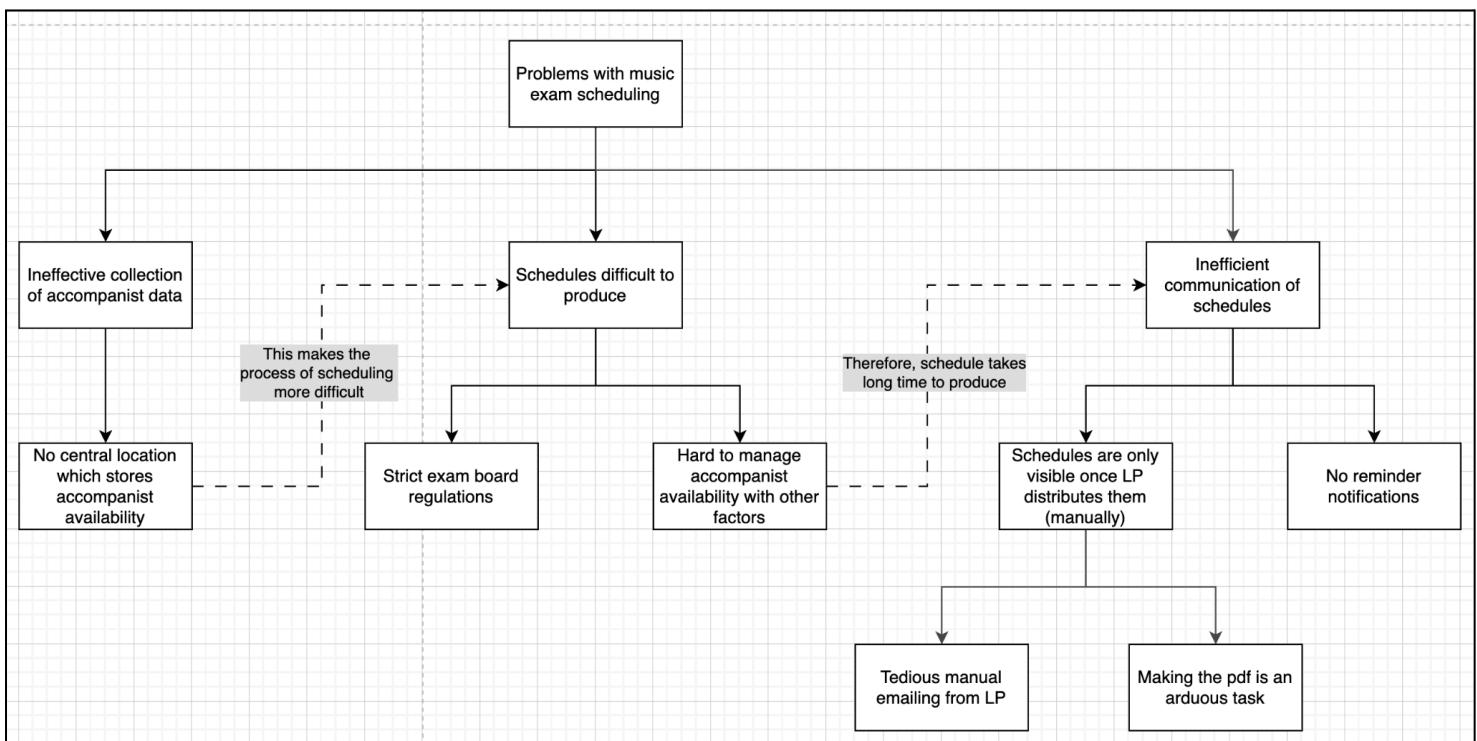


Figure 2.1a: Decomposition diagram of the problem.

This diagram illustrates the key issues with the current method of music exam scheduling, gathered by my research and stakeholder interviews during the analysis stage. The diagram shows 3 key problems:

- Ineffective collection of accompanist data.** Currently, Ms Pearcey talks to accompanists and makes a note of their availability. These notes are not as organised and secure as data in a server would be. Verbal communication can often lead to misunderstandings, which ultimately leads to poorly made schedules.
- Schedules are difficult to produce.** This is the key issue, as the TCL exam board has highly specific rules and regulations regarding the scheduling of these exams. The rules ensure examiners have sufficient break time, and each examinee has a timeslot of a duration specific to their instrument and grade. Furthermore, Ms Pearcey must assign the relevant students with accompanists, each of whom has schedules that Ms Pearcey must work with. She also tries to group accompanists together to save their time. After managing the requirements of accompanists and

the exam board, she also has to deal with any issues raised by students. All of these factors are incredibly difficult to manage simultaneously. This can lead to revised versions, schedules being changed, therefore causing confusion for examinees and accompanists alike. A computer, however, is able to store many variables and make complex decisions using predetermined logic, giving an accurate result consistently. Ms Pearcey herself believes that a computer may not be perfect, saying “It's kind of got to be done like that” in her interview. I could have ignored this issue, but computer software can definitely prove to be a useful tool in the process of exam scheduling, making Ms Pearcey's life easier.

- ***Misunderstandings and unproductive scheduling leads to the inefficient communication of these schedules.*** Mistakes cause delays in the release of the schedule, giving students less time to flag up any concerns (recognised as an issue in [1.3.2 Recent examinee survey](#)). It can also cause confusion, resulting in examinees late to exams, or missing them altogether. The stakeholder interviews also highlighted the importance of succinct and clear communication, especially with younger, more forgetful students. Ms Pearcey wastes her time and energy by manually making a PDF of the schedule, and by emailing the schedule to relevant people. This mundane task deters her from sending reminder emails to students. Automating the process of sending emails and making PDFs is quite simple and effective for computer programs. This problem can clearly be solved by computational methods, which is why I aim to develop its solution.

One could argue that the lack of reminder emails is not a big issue, and hence my decomposition could have taken a different approach. Most sensible students will remember their exams and not need that email. However, it has been recognised by Cyrus and Darius, as well as a large proportion of recent examinees in their survey, that reminder emails would be useful. For younger students who may forget, or for nervous examinees who want to double-check, the reminder email is an important feature to include in my solution. Due to the simplicity of automated, scheduled emails, this task is not too difficult either. Therefore, I have identified this issue as one which I hope to solve with a computational approach.

I have decided to decompose my project by feature, not by user. Therefore, my development stages will be divided into the creation of these features. The main purpose of my project is to schedule, and my main client is Ms Pearcey. All other stakeholders will be benefitted by my project to a lesser extent, and will interact with the software much less than Ms Pearcey. All users complete fundamentally different tasks: the organiser creates and publishes schedules, the other users view timeslots, accompanists enter their availability. This suggests that dividing the problem by its users could be useful. However, I feel that decomposing my project by user will distract from the purpose of this project. Therefore, I believe that my problem, and my solution, should be divided by features. This allows me to focus on Ms Pearcey and her requirements.

## 2.2 Structure of solution

### 2.2.1 Structure diagram

In the previous section, 3 key problems were identified. This lends itself to the solution having 3 major modules; 3 improvements to the current issues.

Key problem	Key solution
Ineffective collection of accompanist data	Gather all inputs and store in a database
Schedules difficult to produce	Make an efficient schedule
Poor communication of schedules	Communicate schedule online, automatically publishing it in several ways

Figure 2.2.1a: Table outlining key solutions to the key problems.

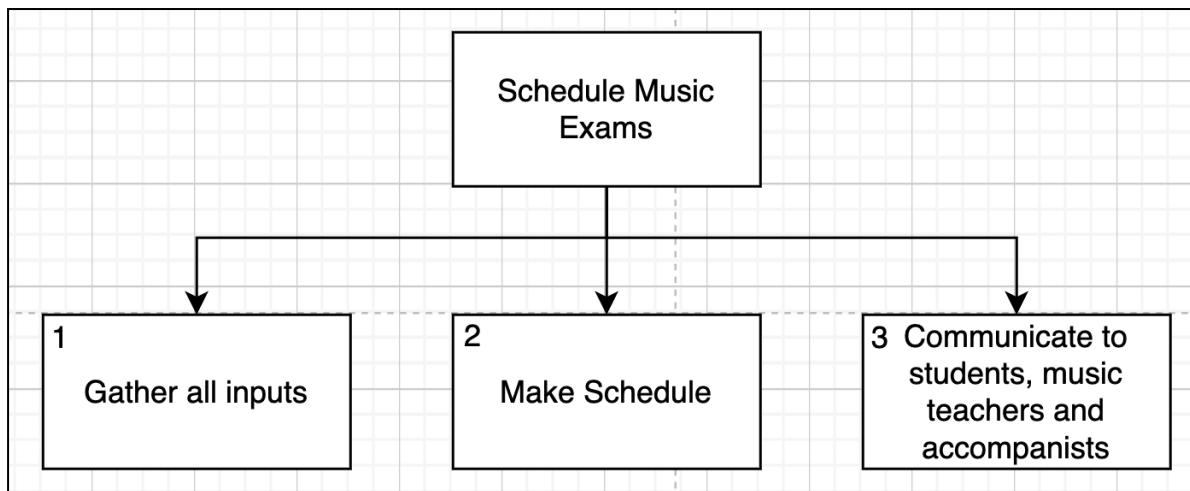


Figure 2.2.1b: Basic structure diagram of the solution

Each of these key solutions have been broken down further into smaller solutions, as shown below.

**Please note:** I am making a key distinction throughout my project between students and examinees. Students are registered users in the system, whereas examinees have signed up for a specific exam. Every examinee is a student, but not all students are examinees. This difference prevents students not taking exams in the current cycle from being disturbed, and prevents complicated many-to-many relationships between different entities ([2.4 Data](#)). For example, a student may play multiple instruments and have many teachers, but each examinee only plays one instrument (as that is the instrument for which they are taking the exam). This also simplifies the process of clearing data for a new exam cycle. Deleting the entity for examinees but not for students keeps as much data as possible. This prevents users having to enter lots of data every time they try to sit an exam, which can become quite frustrating

It is also important to note that accompanists and music teachers have been given the common term ‘supervisors’. This is because one person can be both an accompanist and a music teacher, so they will be stored together in a single entity called ‘Supervisors’. However, each exam cycle will lead to clarification on who the accompanists will be, and this data along with availability information will be stored in the ‘ExamAccomps’ entity. See [2.4 Data](#) for more clarification. Although it may seem confusing, this method ensures that all databases are normalised to third normal form, and an organiser has to enter the least amount of data at the start of each exam cycle.

Finally, “the organiser” is Ms Pearcey, and her initials are LP. These terms are used interchangeably throughout the document, they all mean the same thing.

Before an exam cycle	During an exam cycle
Students	Examinees (they must be a registered student beforehand)
Supervisors	ExamAccomps (they must be a registered supervisor beforehand)

Figure 2.2.1c: Clarification of terms used for different users

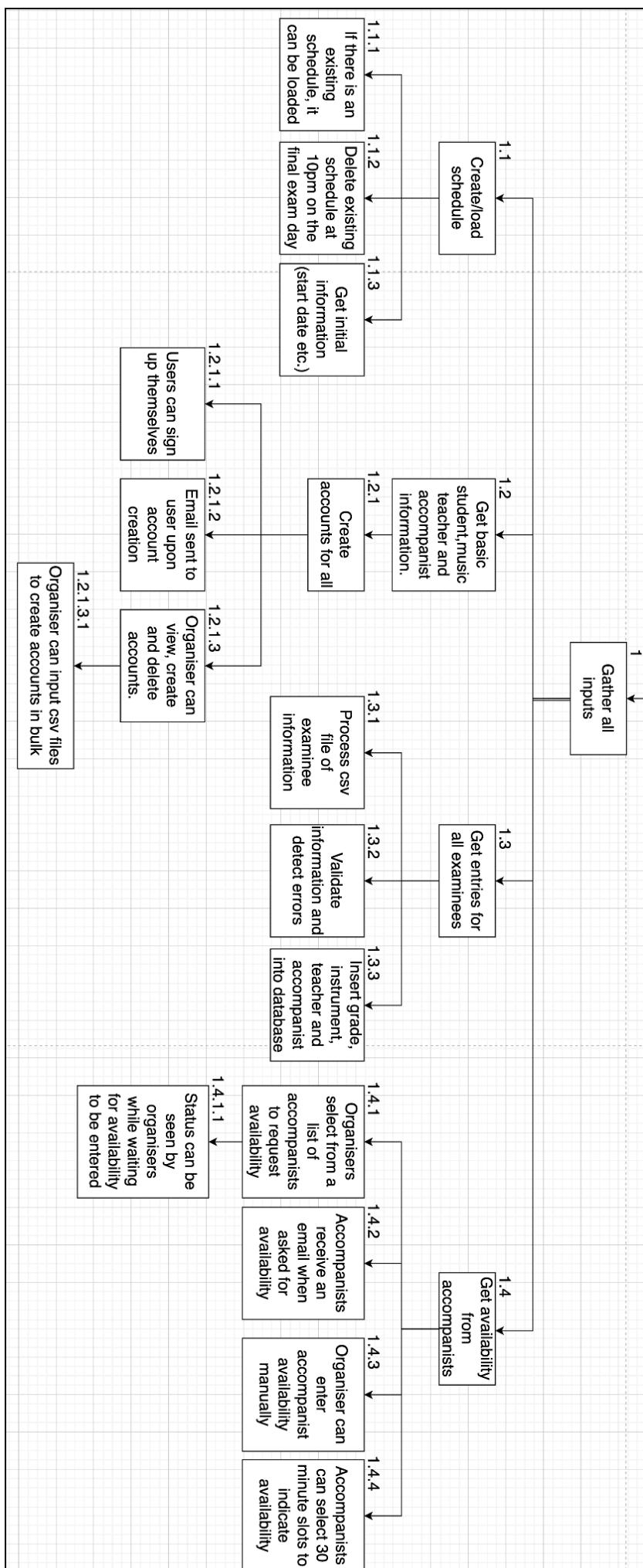


Figure 2.2.1d: Structure Diagram (1) - Gather All Inputs

### *Gather All Inputs*

I intend to develop a system which allows inputs to be gathered and organised efficiently. This involves proper database design to store these inputs, so the user has to enter as little information as possible. These inputs must be validated, giving helpful error messages for any situation that may arise, and it must be easy for the user to enter these inputs. Therefore, deliberate usability features will be included to prevent the user from feeling overwhelmed or confused, but instead to feel comfortable with the software.

**Structure diagram 1.2:** This system has 4 main users: the organiser (Ms Pearcey), students, accompanists, and music teachers. Ms Pearcey is also an accompanist and music teacher herself, and her email will be recognised as the 'admin' login to give her access to the organising functionality. This has been clarified further in [2.4 Data](#). Basic data for students, accompanists and music teachers must be collected initially. One way of doing this is by incorporating accounts into this project. This is a relatively simple method of gathering initial (mostly static) data such as email addresses and names. This is one reason for my decision to use logins, and more reasons have been described later in this section. When students, accompanists, and music teachers register themselves, their details will be sent to the database so that the data associated with each exam cycle is kept to a minimum. This data will be static, so it would not be efficient to have to enter the same information repeatedly with each exam cycle.

Users can register themselves. However, as students can often be forgetful and immature, there will be the opportunity for Ms Pearcey to create accounts in bulk by providing a CSV file of the relevant data for each student. She will also be able to view a page where all accounts can be seen, with the opportunity to amend, or delete, any accounts she needs. This gives her the ability to prevent irresponsible students from being able to access the system, which gives more certainty to the validity of data in the database. It also will allow her to fix any errors that may happen during registration, such as misspelt names. Every time a user is registered, an email will be sent to the provided email address to inform them and introduce them to the system. This prepares them for any future emails that contain more important information, such as exam timings. All of this data will be validated and sanitised to prevent SQL injection, and prevent unexpected inputs into the system. It is important that any invalid data is caught early, before it causes more problems later on. See section [2.4](#) for more details.

**Structure diagram 1.1:** At the start of each exam cycle, the start date will be entered. Then, accompanist availability followed by examinee information is entered to create the schedule. After the schedule is made, Ms Pearcey will be able to view it, and will also be able to re-edit and re-publish the schedule if needed. Once the exams are complete, the schedule and all data associated with that exam cycle has no purpose, so it will be deleted at 10pm on the last exam day. This prevents too much data from being stored in the database, which could reduce costs for the school in the future. It also makes space for the next exam cycle.

**Structure diagram 1.4:** Once everyone has been registered, and a new exam cycle started, the system must gather inputs for accompanist availability. The organiser will select names

from a list, this list will be all of the registered supervisors from the database. The selected accompanists will be emailed, asking them to visit the website to submit availability. At this point, they will view the day from 9am to 5pm in 30-minute blocks, and will click the relevant blocks to show when they are available and unavailable. The blocks will initially all be selected, and the teachers will deselect them to show when they are unavailable. This will encourage teachers to be available more, making the scheduling process easier. They will also be asked if they are the ‘school’ accompanist. As mentioned in section [1.3.2](#), the school accompanist is one person who accompanies the majority of exams. All other examinees will need to be accompanied by a specific person, or will not need to be accompanied at all. Therefore, it is critical that the school accompanist is identified at this point, so the database is able to correctly link each examinee to their accompanist. During this time, Ms Pearcey will be able to view the status of each accompanist that has been requested to enter availability. This will involve viewing the current status of the ‘ExamAccomps’ table (see [2.4 Data](#)). She can then click ‘finish’ to proceed to the next section. She can also choose to enter accompanist availability herself, by individually selecting each accompanist and entering data in the same way as defined above.

**Structure diagram 1.3:** Examinee information is then submitted by providing a CSV file of information for all accompanists. This file must conform to specific regulations, and if it does and all data is valid, it is entered into the database. Through efficient and meticulous database design, Ms Pearcey will have to enter as little data as possible to simplify this process, reduce the possibility of errors and reduce her workload.

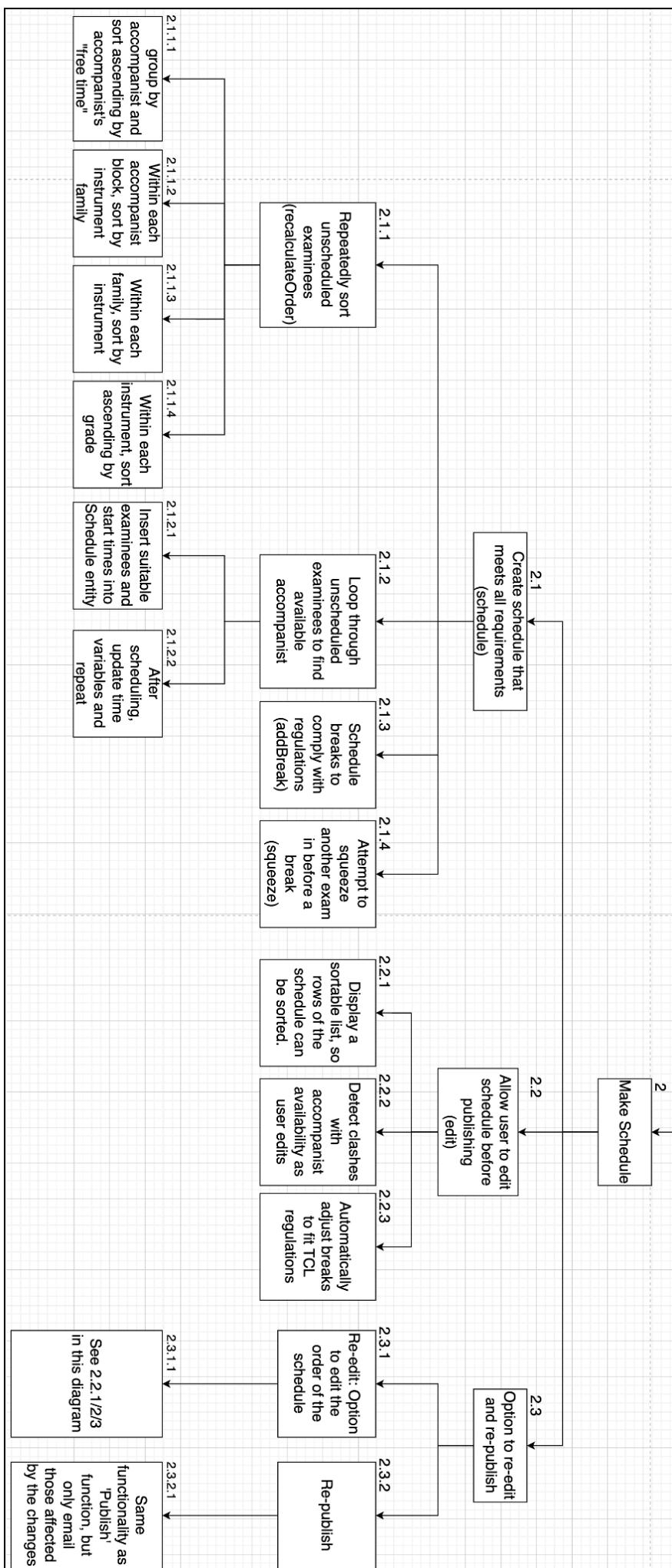


Figure 2.2.1e: Structure Diagram (2) - Make Schedule

## Make Schedule

**Structure diagram 2.1:** This section of the structure diagram is the most important section, and is key to the success of this project. Given valid inputs from section 1 in the structure diagram, this section involves taking the inputs to produce a schedule. The schedule must adhere to regulations outlined in [1.1.3 Updated TCL regulations](#). The regulations, summarised, are:

- The schedule must meet TCL requirements
- Each accompanist's exams must work with their availability
- As many examinees should be scheduled as possible
- Exams should be grouped into accompanist blocks. Within each block, they should be grouped by instrument family, then instrument, then sorted by grade (ascending)

The key idea is that we should try to group exams as explained above, but if we ever encounter a situation where either of the first two rules are broken, go back and try something else. This is the basic principle for the 'schedule' function and how I have designed it.

**Structure diagram 2.1.1:** This block will be developed as a function called 'recalculateOrder'. Firstly, it groups all unscheduled examinees by accompanist. Then, it sorts each group by how much free time the accompanist has. This free time is defined as:

$$\text{Free time} = \text{available time} - \text{exam time}$$

Each block will be sorted (ascending) by the accompanist's free time, so that the most urgent exams are earlier in the list. This means that if 2 accompanists are available at a certain time, the one with less free time (eg. more urgency) will be scheduled. This method attempts to schedule exams so as many exams can be scheduled, and minimal time is wasted or used incorrectly. Within each block, exams are sorted by instrument family, then instrument, then grade. This function is called at the start of scheduling, and it is also called whenever the 'time' variable changes. As exams are scheduled, the 'time' variable is incremented, and an accompanist's free time will be adjusted accordingly. Importantly, 'free time' is **not** a constant, it varies as time changes. If an accompanist's exam is scheduled, 'exam time' decreases (increasing free time and therefore decreasing priority). If an accompanist is available at a certain time but a different exam is scheduled, 'available time' decreases (decreasing free time and therefore increasing priority). The lower an accompanist's free time is, the closer their block of exams will be to the start of the list.

**Structure diagram 2.1.2:** At the start of the 'schedule' function, 'recalculateOrder' is run. This returns examinees in order of their priority for scheduling. By looping through this list and checking if an examinee's accompanist is available, the algorithm makes sure that the ideal examinee (the examinee with the most urgency to be scheduled) *is scheduled*, as long as their accompanist is available.

Once an available accompanist is found, the algorithm checks if the end time of this exam raises any concerns. If it doesn't, this examinee and their start time is then added to the table

'Schedule', the examinee is removed from the list of unscheduled examinees, and the time is incremented accordingly.

However, if concerns are raised the algorithm cannot schedule the exam. These concerns involve checking if the end time will break TCL regulations in any way, as well as checking if it is time for a break (as according to figure 1.1.3a in [1.1.3 Updated TCL regulations](#)). The critical idea used here is to realise that all of the TCL regulations are centred around when a break is scheduled. If the TCL regulations are broken in any way, it is either time for a break or it is the end of the day. Therefore, if concerns are raised, and there is no time to squeeze in one more exam, a break is added (addBreak - **structure diagram 2.1.3**). Please note that 'addBreak' will schedule a 15 or 60 minute break appropriately, and will also detect if it is the end of the day and time to start the next day. When scheduling a break, the main change is that time is incremented by 15/60 minutes. The Schedule table is not affected. If concerns are raised, and the algorithm thinks that one more exam can be scheduled, the 'squeeze' function is run.

**Structure diagram 2.1.4:** The squeeze function will calculate the difference between the current start time of an exam, and the ideal break time. This difference is the ideal duration of an exam. It then searches through unscheduled examinees with no accompanist, and unscheduled examinees with the accompanist that was about to be scheduled. It looks for the closest match to this ideal duration. The exam with the closest match is checked to see if any concerns are raised. If it meets all regulations, then it is scheduled. However, if concerns are raised by this new exam, 'addBreak' is called to add a break. This tries to schedule as many exams as possible, but does not force accompanists to come in and play just one exam too far outside their group. The accompanist about to be scheduled is likely to already be accompanying exams, or is about to start doing so, therefore they will not be affected by this exam time.

**Structure diagram 2.2:** After a schedule is made, the organiser will be able to either edit and then publish, or publish straight away. If edit is clicked, the schedule is displayed as rows that can be moved by the user. Every time a movement occurs, the 'edit' function will recalculate start times and breaks, and automatically raise any clashes with accompanist availability. This allows the user to adjust the schedule as they like without breaking TCL regulations, but does allow them to schedule exams when their accompanist is unavailable. In this case, it will display a large message to inform them of this issue, but does not enforce anything. I have included this feature after further discussion with Ms Pearcey about this process. She informed me that she sometimes goes back to an accompanist and asks if they can play at a time just outside their availability, and they usually agree. For whatever reason, Ms Pearcey may find herself in a similar situation, so I decided that it is crucial for the program to support this. Therefore, the system will display a message to say "this accompanist is unavailable", but if Ms Pearcey knows better she can ignore the message and publish it anyway. This method provides the most information and advice without forcing anything to happen, which is something I liked when researching Furlong Maestro (section [1.4.3 Furlong Maestro](#)).

However, the TCL regulations will always be fixed, so they are used to automatically update break times with each movement. When a movement is made, the schedule is looped in its new order, and starting from 9am each exam's duration is used to change the start time. If an exam's new time is raising any concerns (is this exam breaking TCL rules / is it time for a

break), a break is scheduled by incrementing the time accordingly (addBreak - structure diagram 2.1.3). During this loop, each exam's accompanist availability is checked. If the accompanist is unavailable during this exam, or during part of the exam, a message is displayed on screen. The new schedule order and new times will then be changed in the database and displayed on screen.

**Structure diagram 2.3:** After Ms Pearcey has edited the schedule until she is happy, she clicks 'publish' to send the schedule out to everyone, as well as carry out other functions (see structure diagram 3). If she logs on again, after publishing the schedule, she will be able to view the schedule. There will also be a function to re-edit the schedule, taking her back to structure diagram 2.2. I could have allowed her to edit the accompanists and examinee information too, however I decided against it. She stated in our stakeholder interview that:

*"The only thing that would then be an issue once I've published this out would be if somebody turns around to me and says, oh actually, I've got a trip that day or I've got a dental appointment. And then I have to then look at juggling things around but at that point, I'd be trying to juggle within the blocks that I've got."*

(see full interview in [Appendix](#))

As the most common solution is to 'juggle' the order of the exams, I found that there was no need to make the system any more complicated and potentially confusing for the user. Although this system will help Ms Pearcey considerably, and aims to solve the problem of exam scheduling, there are already parts of her method which are efficient and reliable. Therefore, it is crucial that Ms Pearcey is able to carry out these tasks with the new system. This allows her to feel more comfortable with the program and have more control over what can happen, while also being assisted by the technology of calculating breaks and detecting clashes.

Once she is happy with the changes made, Ms Pearcey can click 're-publish'. This feature will update the schedule for everyone, and work in the same way as the 'publish' function (see structure diagram 3), but with one major change. Only those affected by the change will be emailed the new timetable. This prevents users from receiving duplicate emails which could confuse them, while still informing all the users who need to know about these changes. The re-publish function will iterate through the examinee field in the old schedule and new schedule using count-controlled iteration, and check if the current exam has the same examinee across both schedules. If true, the time slot has not changed and this examinee does not have to be emailed. If false, they are added to a list of examinees to be emailed. Once the whole schedule has been checked, this list is submitted as a parameter in the 'publish' function. This will lead to the appropriate students, accompanists and music teachers being emailed.

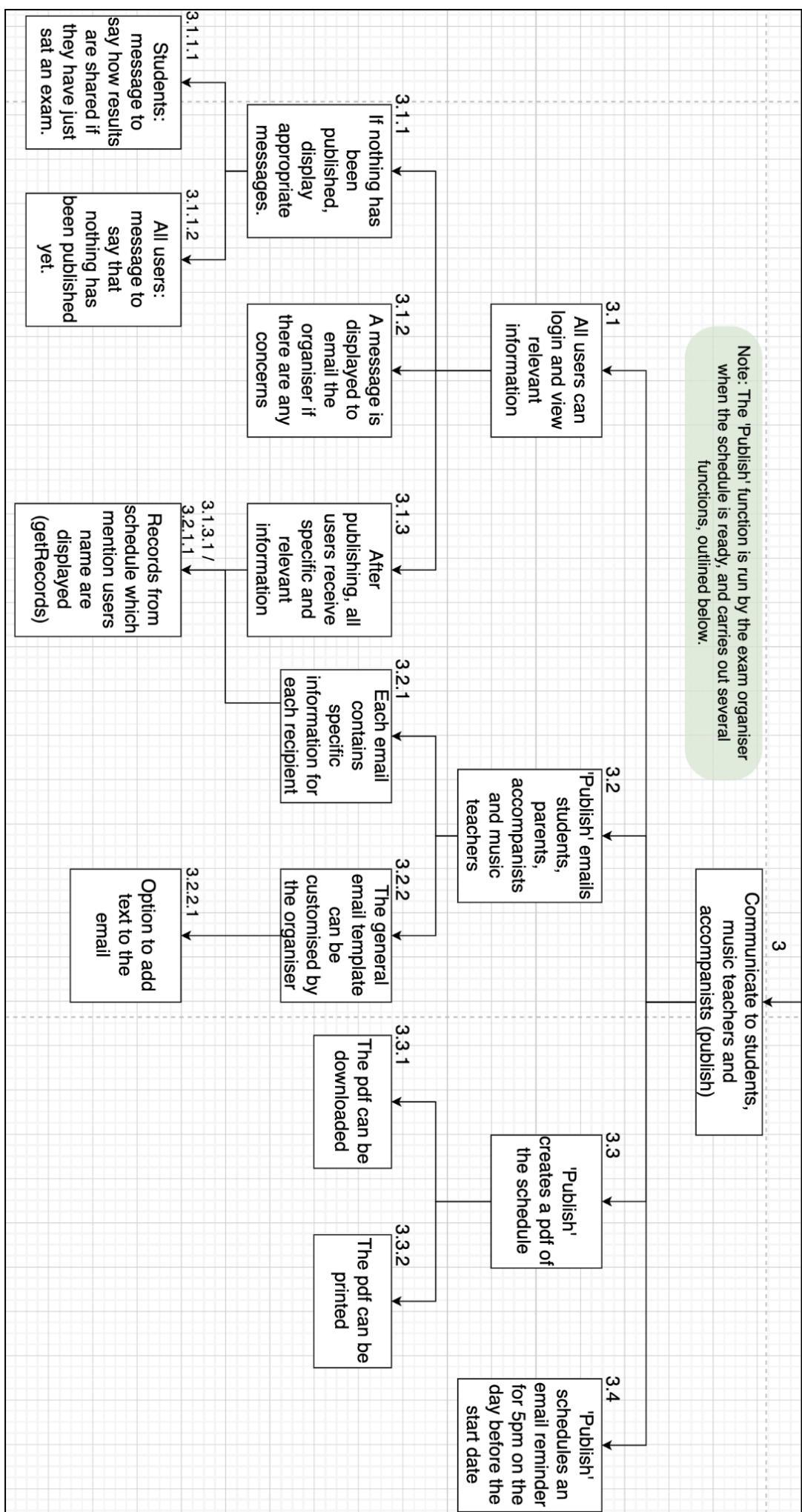


Figure 2.2.1f: Structure Diagram (3) - The ‘publish’ function

### *Publish Schedule*

The final part of my structure diagram represents the final key solution outlined in figure 2.2.1a. This section aims to solve the issues Ms Pearcey currently faces when distributing the schedule. These problems include: manually emailing everyone, manually creating a PDF file, taking a long time to distribute the schedule, and not sending reminder emails. Each section of **structure diagram 3** is designed to mitigate one of these issues.

**Structure diagram 3.1:** As mentioned previously, a benefit of implementing user logins is that it allows data to be collected easily to inform the scheduling process. However, the main benefit of this feature is its ability to send personalised information to each user, and to ensure that only certain users can access certain features. Once the user logs in, they are taken to their dashboard. Here, any information that is relevant to them will be displayed. If nothing has been scheduled yet (the organiser has not clicked ‘publish’), this will be displayed on the dashboard. This message will aim to relax any nervous students and inform all users about this system, something similar to: “The upcoming exams have not been scheduled yet, but as soon as they are, you’ll get an email and this page will be updated.” This message will be displayed to all users, but students will receive an additional message. This message will say “If you have recently taken an exam, results will be displayed on the music notice board, or you will be contacted by your music teacher.” This lets any recent examinees know how they will find out the results of their exam, and prevents them from thinking it will be available here.

It is important to note that ‘nothing being scheduled’ means that, if the table exists, “IsPublished” is set to false in the BasicInfo table (see [2.4 Data](#)). At the beginning of an exam cycle, this field is false and ‘nothing has been scheduled’. Once published, it is true. However after the exam cycle ends, the table is deleted. In this scenario, we are technically at the beginning of a new exam cycle and therefore ‘nothing has been scheduled’. If an examinee who has just sat their exam logs into the system, they may expect to see their results. However, they will only see a message saying ‘nothing has been scheduled’. This can be confusing for examinees who are looking to find out their results, so the additional message is designed to clarify any confusion.

In any situation, the dashboard will always display Ms Pearcey’s email saying to email her if the user has any concerns. The reason for this feature is after my recent examinee survey (section [1.3.2](#)) revealed that a considerable number of examinees had an issue with their time slot but did not act on it. In order to prevent this from occurring again, I designed this message to make the user feel encouraged to inform Ms Pearcey of any problems. This will lead to students feeling more appreciated and happy with their timetable, reducing stress and leading to improved exam performance on the day.

Once published, “IsPublished” is set to true. This causes an algorithm (getRecords) to display record(s) from the schedule which mention the user’s name. Therefore, the user will be able to see the time slot(s) relevant to them, abstracting the rest of the data which may confuse them. This algorithm will execute SQL queries to find time slots of the schedule where the user’s name is mentioned. For students, this query will be relatively simple:

SELECT \* FROM Schedule WHERE ExamineeID = \$userID. For supervisors, as one user can be an accompanist and a music teacher, this will involve making 2 separate queries. As the database is normalised to 3rd normal form, supervisor information will not be stored in the Schedule table. The accompanist and music teacher depends entirely on the examinee, so including examineeID **and** supervisor information in the 'Schedule' table would introduce transitive dependencies. This has been explained clearly in [2.4 Data](#). As a result, accompanist and supervisor IDs will require SQL's JOIN function to be used, because they are stored in other tables.

**Structure diagram 3.2:** The same function explained above (getRecords) is used to send specific emails to each user. The 'publish' function automates this process by sending all emails as soon as the function is run. Using the PHPMailer library, I intend to send emails to each user. This email will contain basic text like "Ms Pearcey has published a new schedule". Then, it will contain information that is specific to the recipient using the function outlined above. Finally, some text will be displayed at the bottom of the email. This text will be entered by the organiser after clicking 'publish'. There is no requirement to enter any information, but I believe that it is important to allow some level of customization into the email. The reason for including this feature is after I noticed some additional text at the bottom of a printed schedule in my school's music exam board.

Tuesday 11<sup>th</sup> July 2023

Forename	Surname	Instrument	Grade	Time
Diya	Sharath	Classical Guitar	1	9:00
Sara	Sadiq	Classical Guitar	1	9:13
Zaynah	Jamal	Classical Guitar	2	9:26
Charde	Levermore	Classical Guitar	3	9:39
Reena	Koiri	Classical Guitar	3	9:52
Olivia	Yang	Classical Guitar	3	10:05
Poppy	Willis	Classical Guitar	3	10:18

13/06/2023 Version 1

Examiner: TBC

You will need to arrive *at least 15 minutes before* the time shown on this notice.

Please advise Mr Watters or Miss Hawthorne *immediately* of any problems.

The exams will be held in the Boys' School Music Block

Figure 2.2.1g: Additional text at the bottom of a printed schedule. This has been cropped from figure 1.5.3a, which displays the full notice board.

This image shows that Ms Pearcey wanted to send additional information to everyone. Therefore, I have included the feature of adding text to everyone's email. I did not want to make the customizability of the email too complicated, as this would go against the simplicity of the 'publish' function. Something which stood out to me when researching Practice Pal (see [1.4.2 Practice Pal](#)) is the simplicity of a very similar function, and this is something I hope to emulate with my solution, by minimising the amount of user inputs after clicking 'publish', and automating several processes with that single function. I believe that this

feature maintains simplicity while allowing Ms Pearcey to express enough information to the recipients of the email, so I have not designed any more customisation features.

**Structure diagram 3.4:** Once again, PHPMailer and the getRecords function are used to schedule a reminder email. In [1.3.2 Stakeholder interviews](#), I received large support for the idea of a reminder email. Therefore, all users will receive one at 5pm on the day before the first exam. The algorithm to carry this out will work in a very similar fashion to the original email specified in structure diagram 3.2, but this time the organiser will not be able to input any more information. It will contain outputs of the getRecords function, so recipients only view information relevant to them.

**Structure diagram 3.3:** Perhaps the most important output of the schedule is the PDF produced, which is printed and posted on the music notice board. From my recent examinee survey (section [1.3.2](#)), 95% of examinees found out about their exam through the music notice board. While some were also informed by email, and many expressed interest in receiving a reminder email, it is evident that the majority of students and supervisors are comfortable with the music notice board informing them of the schedule. I do not want this system to cause any confusion or disrupt current methods that are working fine. However, I do want to prevent Ms Pearcey from the unnecessary task of producing this PDF. This feature will produce a PDF automatically, so it can be downloaded and printed to be placed on the music notice board. This will allow my new system to streamline (but not disrupt) a process which 95% of examinees are familiar with.

Using the DOMPDF library, I will use HTML and some basic styling in CSS to create a PDF file of the schedule. This will be available to download by clicking a button on the screen. After downloading, Ms Pearcey will be able to distribute the schedule however she likes, which will involve printing it out to put on the music notice board. This library allows me to use PHP to create PDF files, while also using HTML / CSS to structure the file as I like.

Previously made schedules such as those in figure 1.5.3a helped me to decide which fields should be included in the file. However, I was unsure about exactly which fields to use, so I emailed Ms Pearcey to clarify her requirements. We came to an agreement that the file should include the examinee's first name, last name, instrument, grade, accompanist and start time of the exam.

The screenshot shows an email exchange. Rohan Desai (rohan.desai407@camphillboys.bham.sch.uk) sent a message at 17:53 (4 hours ago) to Lorne Pearcey. Rohan asks about what to put in the final PDF of the schedule, suggesting student's first and last name, instrument, grade, accompanist, and start time. Lorne Pearcey (lorne.pearcey@camphillboys.bham.sch.uk) responded at 18:33 (3 hours ago), saying "No. I think that's fine." Both messages have a yellow star icon and a three-dot menu icon.

**Rohan Desai** <17desai407@camphillboys.bham.sch.uk>  
to Lorne ▾  
17:53 (4 hours ago) ☆ ↵ ⋮

Hi miss,

Hope you are doing well. I have a quick question: what should I put in the final PDF of the schedule? Currently I am thinking of putting the student's first and last name, instrument, grade, accompanist and start time. Is there anything else to put in?

Thanks,  
Rohan

---

**Lorne Pearcey**  
to me ▾  
18:33 (3 hours ago) ☆ ↵ ⋮

No. I think that's fine.

Ms L. Pearcey

Figure 2.2.1h: Checking requirements of the PDF file with my client.

After clarifying with my client, I was confident to design the following template for the PDF, as an example for how the file should be structured during the development stage.

# TCL Exam Timetable

20th July

Examinee First Name	Examinee Last Name	Instrument	Grade	Accompanist First Name	Accompanist Last Name	Start Time
Adnan	Husayn	Violin	1	Simon	Palmer	09:00
Amogh	Shetty	Violin	2	Simon	Palmer	09:13
Prasanna	Sivakumar	Violin	2	Simon	Palmer	09:26
Ishaan	Dubey	Clarinet	1	John	Doe	09:39
Rohan	Desai	Drum Kit	8			09:52
...	...	...	...	...	...	...
...	...	...	...	...	...	...
...	...	...	...	...	...	...
...	...	...	...	...	...	...
...	...	...	...	...	...	...
...	...	...	...	...	...	...
...	...	...	...	...	...	...

21st July

Examinee First Name	Examinee Last Name	Instrument	Grade	Accompanist First Name	Accompanist Last Name	Start Time
Simon	Liu	Drum Kit	5			09:00
Yijun	Chen	Saxophone	3	Simon	Palmer	09:23
...	...	...	...	...	...	...

Created on: DD/MM/YYYY

Figure 2.2.1i: Template for PDF of timetable, which will be created by the DOMPDF library.

The title will remain constant: 'TCL Exam Timetable'. The start date will be displayed clearly, then a table will follow containing all relevant information, as specified earlier. The start time of each exam should be in bold, to highlight the most important information. This draws the reader's attention, minimising the risk of forgetting the exam. This is repeated for each day the exams run for. At the bottom of each page, the current date (day when the schedule is published) is displayed. This allows the organiser to differentiate between schedules if they happen to re-publish a schedule.

The relevant data will be extracted from the database using SQL queries. By joining data from multiple tables to the 'Schedule' table (using SQL's JOIN function), I can create the desired row to enter into the file. To highlight breaks, the algorithm will check if the next exam's start time is equal to the current exam's start time + duration. If they are not equal, a break is scheduled between these exams, so the algorithm will print a blank row in the file. This allows accompanist's to see when they may get a break, and divides the document into sessions, making it easier to read.

This structure diagram clearly shows that each of the 3 key solutions can be decomposed into smaller solutions. The structure of my entire solution has clearly been defined and clarified in this section. Together, each module of the structure diagram can be developed to produce a full, effective solution to this problem. My design and development will take a modular approach, allowing me to work in a logical manner. By rigorously testing each module, and then testing the larger solution, I can work towards a successful final product.

## 2.2.2 Swimlane Diagram

I have produced a swimlane diagram of the account creation process, and a larger swimlane diagram for the subsequent exam scheduling process. This helps me to understand in which order each component will be run, and how they can come together to give a full solution. This clarification is vital to understand how each module of my solution can be structured and developed, as I cannot develop the solution without a deep understanding of how each module will work together, and which modules are more crucial to the success of the solution. It also helps me to understand what parts of the process each user is involved in, which helps me to develop features catered to these users.

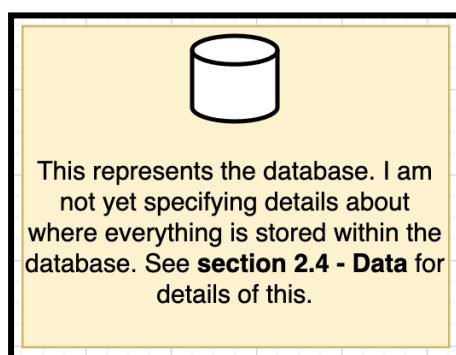


Figure 2.2.2a: Initial disclaimer about swimlane diagram

As figure 2.2.2a says, all details about the structure of the database are in section [2.4 Data](#). The database symbol in this diagram simply indicates when something is stored in the

database. The description in this section may start to introduce features of the database, but section 2.4 is where this is fully explained and clarified.

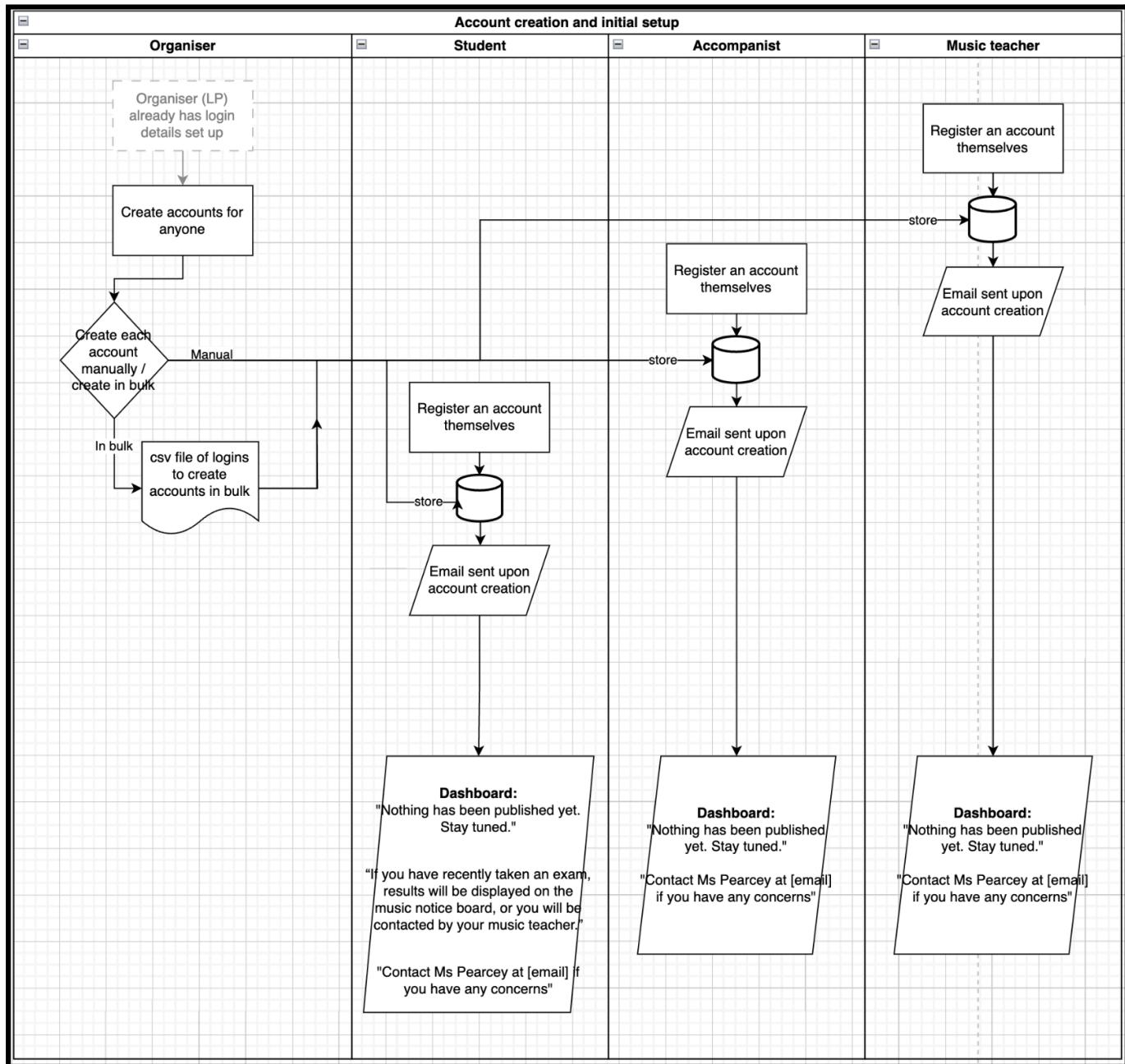


Figure 2.2.2b: Full swimlane diagram of account creation process

### *Account Creation*

To gather initial data for all users, and store it appropriately in the database, all users must register an account. This will allow their email to be linked to their StudentID or SupervisorID, so when examinee information is submitted, a simple email address will be enough to uniquely identify each user. Initially, I planned on students having to enter music teacher information during registration. However, that means that all music teachers would have to be registered beforehand, which could cause confusion and require more effort for Ms Pearcey to coordinate. It also prevents students from having multiple music teachers, and

therefore students who play multiple instruments are not supported. Therefore, I decided to include music teacher information when inputting the examinee information. This means that one student has one teacher **per cycle**. So, in the next cycle, they can choose their other music teacher (if they are taking an exam for a different instrument). This supports many to many relationships between teachers and students without creating these relationships in the database. This does mean that one student can only sit one exam per cycle, but after discussing with Ms Pearcey this is incredibly rare, so not a concern. This change also allows any users to register at any time, simplifying Ms Pearcey's workload when informing people about this new system.

Ms Pearcey will also be able to create logins in bulk by submitting a CSV file, if she prefers this method. As students and supervisors have separate tables in the database (and they are required to enter slightly different data during registration), Ms Pearcey would have to submit a CSV of student information, followed by a CSV of supervisor information. This data includes name, email and parent email. All of this information is available on school management information systems (MISs) such as SIMS, which is used by my school. These systems allow this data to be exported as CSV, which can directly be imported into this system. This simplicity allows Ms Pearcey to create many accounts very easily, as the common format of CSV is widely supported. Every user created in this way will be able to set their password the first time they log in. All users will receive an email when their account is created, and will be able to access their dashboard. This will display messages similar to what is shown in figure 2.2.2b and structure diagram 3.1. Once all users are registered, Ms Pearcey can start the exam scheduling process.

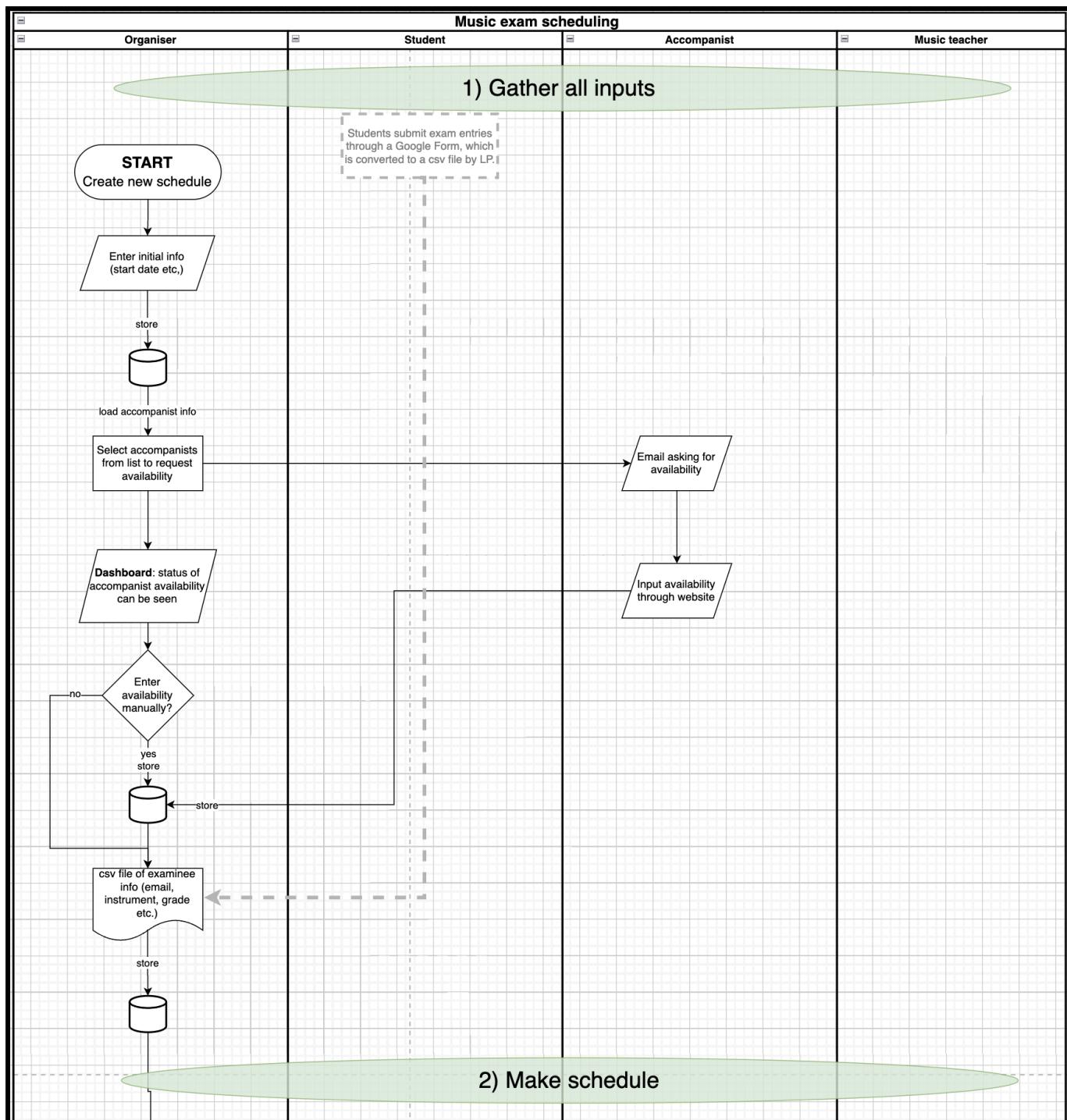


Figure 2.2.2c: Swimlane Diagram of scheduling process (1) - gather inputs

### Gather All Inputs

If nothing is scheduled yet, the organiser's dashboard will display a button to start a new schedule. The first data to enter is the start date of the exams. This will be stored in the BasicInfo table, with 'IsPublished' set to False and the current timestamp recorded. To avoid errors with email scheduling (scheduling emails in the past) and other future errors, the start date will be validated to ensure it is in the future. It will also be stored in a fixed format (ie. DD/MM/YYYY). If the date fails validation checks, an error message will be displayed and the organiser can try again.

The next step is to gather accompanist availability. This is collected before examinee information intentionally. The examinee information will contain accompanist emails, and in order to link these to AccomIDs, the accompanists need to be identified first. If data was collected in the other order, the CSV of examinee data would have to be stored in the server, and only processed once accompanist data is collected. This method would become more complex and waste storage space and power.

Ms Pearcey will select accompanists from a list of registered supervisors, and they will be emailed, asking them to enter availability. They will be directed to their dashboard, where they can click large blocks to enter availability. These blocks will represent 30-minute times from 9am to 5pm. They will also be able to check a box asking if they are the ‘school’ accompanist. This is very important, as it allows all examinees to be correctly linked to their accompanist. The CSV of examinee information will have a field ‘Accompanist’. The possible inputs of this field is: null, the accompanist’s email address, or ‘school’. The string ‘school’ is acting as a variable, representing the one accompanist who is taking on the role of school accompanist. Therefore, it is critical to recognise who the school accompanist is **before** examinee information is submitted. All of this accompanist data will be stored in ‘ExamAccomps’. The SupervisorID associated with them will be stored when they login. This ID will be renamed as ‘AccomID’ in ExamAccomps, to help distinguish between the 2 fields. All remaining fields stored in ExamAccomps are boolean, so no validation is needed. It is worth noting that all fields in the ExamAccomps table can be 0 except the primary key (AccomID), so as soon as Ms Pearcey asks an accompanist for availability, their ID will be entered as a new record in the table. Adding availability will involve altering this record. This allows the ExamAccomps table to reflect the status of **all** accompanists selected, not just those who have submitted availability.

The organiser will be able to view the status of accompanist availability while waiting. This will involve viewing the ExamAccomps table at its current state, so all who have and have not entered their availability yet can be detected. This gives Ms Pearcey the opportunity to remind those who haven’t. At this point, she will also be able to manually enter/ alter availability for each accompanist individually. In this scenario, she will be able to select an accompanist from the ExamAccomps table and alter the record. When clicking on an accompanist to edit, she will be taken to the same page that accompanists are shown. If Ms Pearcey is happy with everyone’s availability, she can click ‘continue’.

Then, a CSV file of examinee information can be submitted. This will include examinee email address, instrument (exact name as shown in [1.1.2 TCL Regulations](#)), grade, music teacher email address, and the accompanist column. As previously mentioned, the accompanist column will either be: null, the accompanist’s email address, or ‘school’. The file will be validated to ensure that all inputs are sensible, and all emails exist in the correct tables. This has been specified in [2.4 Data](#). If there is an error found during validation checks, the error will be displayed on the webpage and Ms Pearcey can try again.

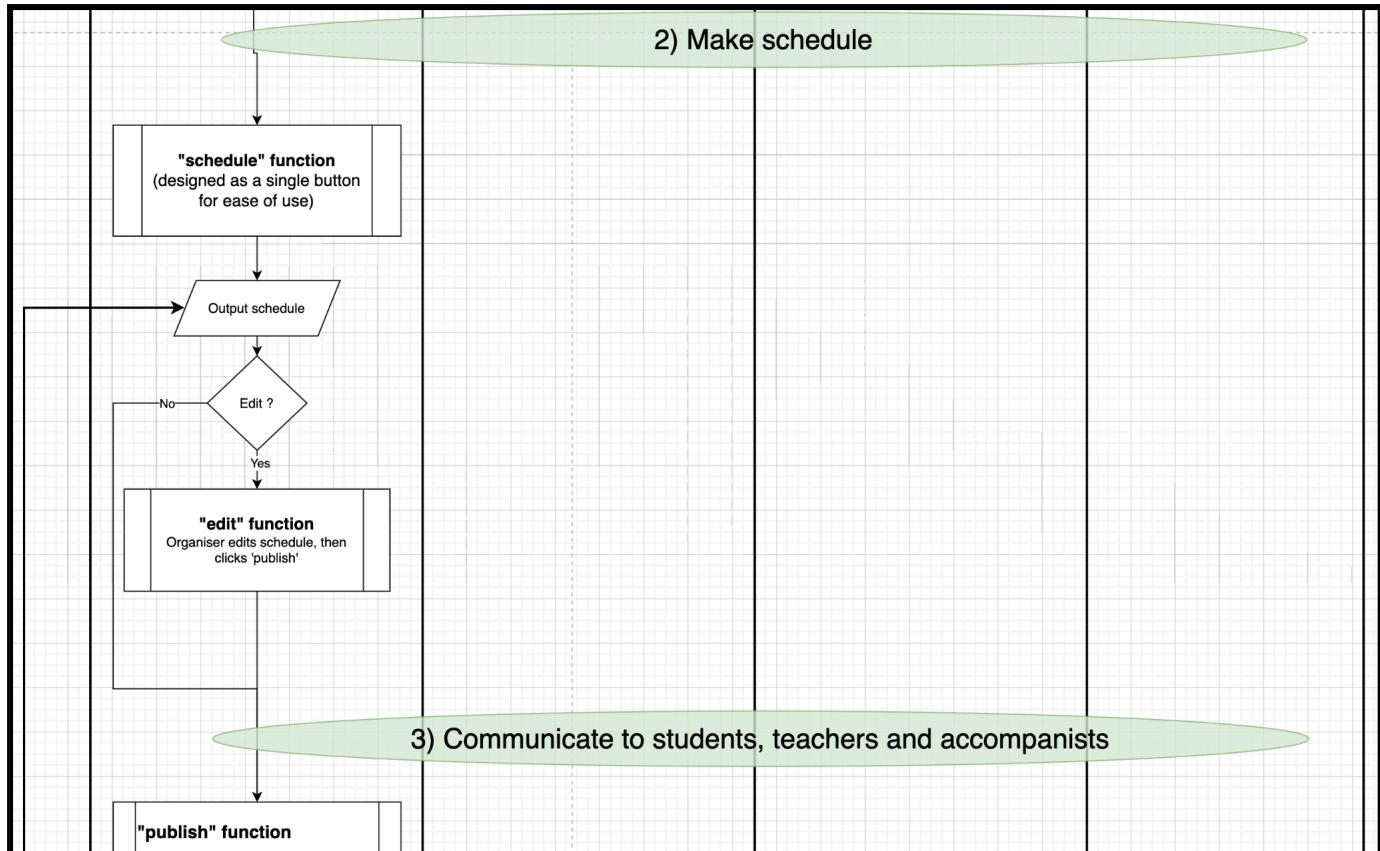


Figure 2.2.2d: Swimlane Diagram of scheduling process (2) - make schedule

### Make Schedule

This section of the diagram is entirely carried out by the organiser. Once all inputs have been validated, Ms Pearcey can proceed to the next step: the schedule function. This function will take all inputs to produce a schedule, and this schedule will be displayed on screen. The schedule function will be designed fully in [2.5 Algorithms](#), with a flowchart representing the intricacies of this algorithm. The schedule will be displayed on screen in a very similar format to the final PDF (figure 2.2.1i). To clarify, it will not produce a PDF at this stage, but the screen will look very similar to figure 2.2.1i. This means that the screen will display the timetable with the fields: examinee's first name, last name, instrument, grade, accompanist's first and last name, and finally the start time of the exam. There will be blank rows in the table to represent breaks, and the table will be split into individual days. Alongside this timetable, 2 buttons will be displayed: 'edit' and 'publish'. If publish is clicked, the publish function will be run immediately. But if 'edit' is clicked, Ms Pearcey has the opportunity to edit the order of exams, and then publish this timetable.

Once the 'edit' button is clicked, the schedule with sortable rows will be displayed. Ms Pearcey will be able to drag and drop these rows to adjust the order of the schedule. Each time she does this, the 'edit' function will recalculate the start times for all exams, as well as recalculating breaks in order to meet TCL regulations. The function will also check if the accompanist is available for each exam, and a message is displayed on screen if they are not. Ms Pearcey can now click 'publish' to publish the schedule with her edits. The 'edit' function will be designed in section [2.5 Algorithms](#).

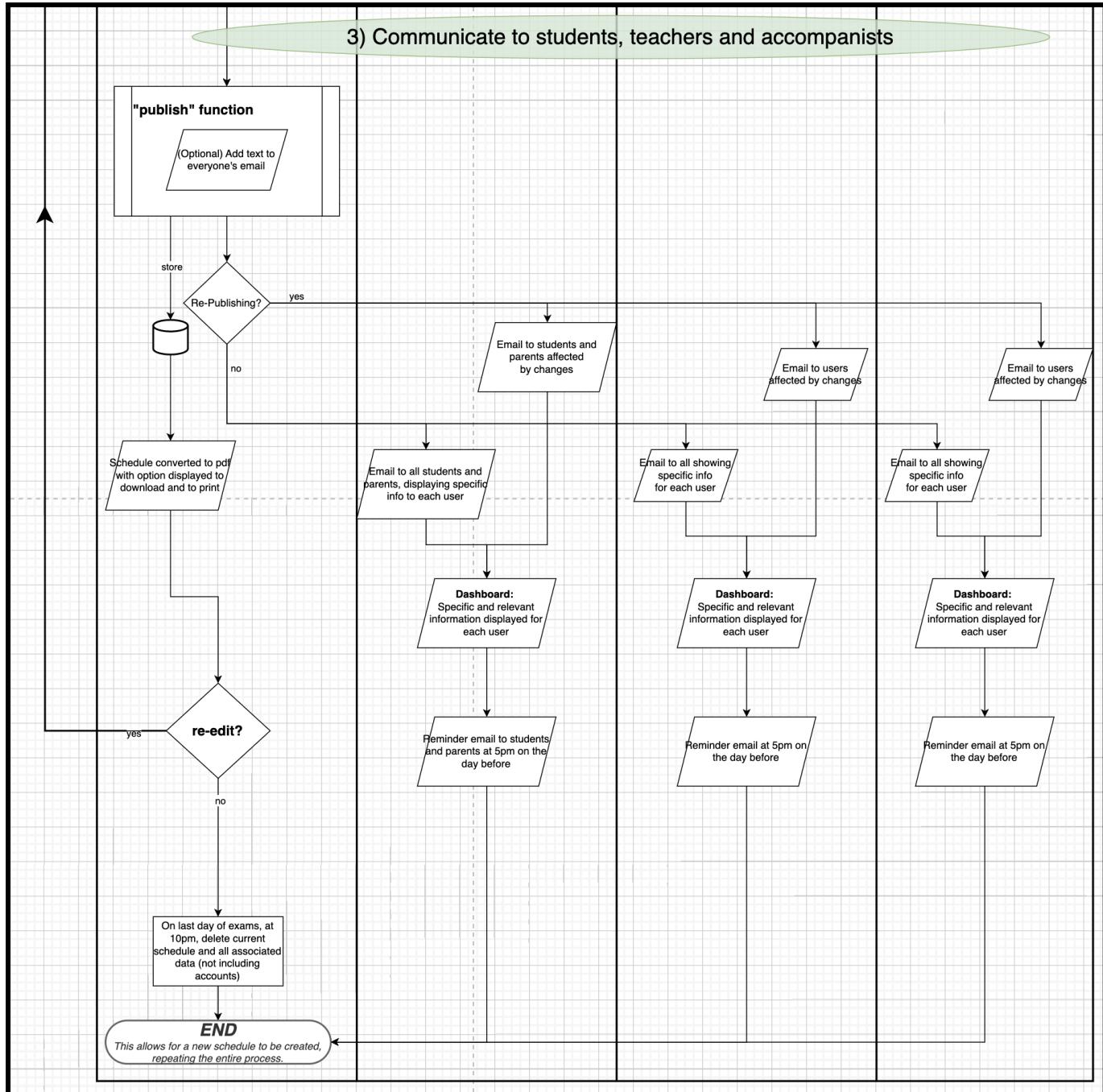


Figure 2.2.2e: Swimlane Diagram of scheduling process (3) - publish

### *Publish Schedule*

Unlike the other sections, this section involves very little user input and a large amount of outputs. The simplicity of this function is intentional, it is designed to have as little input as possible by automating several processes simultaneously. The 'publish' function will be designed in more detail in [2.5 Algorithms](#), so I will not describe and explain the details of this function here. Once Ms Pearcey clicks a button to 'publish', the function will run. Firstly, it sets 'IsPublished' to True, this is found in the BasicInfo table in the database. This will cause the dashboard for each user to display the relevant information specific to each user, abstracting the rest of the schedule to maintain simplicity. Next, it will send emails containing this information to all users. Additionally, parents will receive the same email that students

receive. A reminder email will also be scheduled containing similar information. Finally, the function will use the DOMPDF library to create a PDF of the schedule (more details in figure 2.2.1i). After publish has finished running, all user dashboards will contain relevant information, all emails will be sent/scheduled, and the PDF will be created. When the function is complete, Ms Pearcey's screen will display a message to say this, along with a button to download the PDF and another to print the PDF immediately. If she logs in at a later date, she will be given the option to re-edit and re-publish the schedule. If she does not select this option, nothing changes. If she does, she is taken back to the scheduling process, and can adjust the order of the timetable (this is the arrow that goes from figure 2.2.2e to 2.2.2d). Any examinees that have had their exam time changed are recorded, and the publish function will only email those examinees and their parents, accompanists and teachers. Anyone not affected by the changes will not be emailed to prevent confusion. When re-editing, 'IsPublished' will be set to False (preventing anyone from viewing the schedule). Once the timetable is re-published, it is set to true and the new schedule is published. A new PDF is also created.

At 10pm on the last day of exams, all data associated with the current exam cycle will be deleted. This makes space for a new schedule to be created, and the process is repeated.

As mentioned in section 2.1 of this document, I will be developing my system by feature, not by user. This diagram clearly illustrates the extent to which organisers will use the software compared to other users. Other users register, some enter their availability, and then they view the published schedule. The organiser, however, will be involved throughout the entire process, completing many different tasks and running many functions. Therefore, it seems sensible to develop this project by the features that the organiser has to complete. This also helps me to focus more on my client, Ms Pearcey, who is the organiser.

### 2.2.3 Choice of programming languages

To develop this solution, I am using WampServer. This Windows web development environment allows me to efficiently develop my web application and manage its databases from my computer. This means that I can easily test from each user's perspective, and also view the database status from my computer. In practice, this website would have to be hosted online, most likely through a paid web hosting service. The solution would remain the same, but the program files would simply be stored in a different location.

To develop the web page and user interface, I will be using HTML and CSS. This is suitable for creating websites, and the CSS will be kept as a separate file, allowing for colour coordination and web design to be consistent across multiple pages. To keep the webpage looking professional and clean, I will use Bootstrap 5 to style most of my website. Bootstrap is a front-end web development framework which helps me to create a visually appealing website. I will still use CSS to tweak this styling to achieve specific usability features. The majority of my code will be written in PHP, as my project requires lots of server-side operations as a result of user inputs. PHP allows me to communicate with my database, and process inputs from the user to create these schedules. Part of WampServer's functionality includes phpMyAdmin, an administration tool to improve the efficiency of managing databases with MySQL. At its core, this means that I will use SQL to create and manage my

relational database. SQL is the only language that supports managements of relational databases, so it is a simple choice for my project. I will additionally need to incorporate client-side functionality into my website, so will use JavaScript for this aspect. All of these languages can work together to allow me to create a well-structured and efficient solution. To summarise, I will be using the following languages:

- PHP
- SQL
- JavaScript
- HTML
- CSS

Due to their popularity, there is lots of online support for all of these languages. This helps me to solve any issues that may arise during development. There are also lots of frameworks, libraries and other tools for these languages which can simplify my workload, and allow different languages to communicate with each other. For example, I will be using the MySQLi extension to carry out SQL operations using PHP.

## 2.3 Usability features

I am aiming to design a program that is easy to use, so the user feels comfortable to interact with the software. In order to do this, it is key to understand each user involved and their proficiency with computers:

- Ms Pearcey: The main user and my client. As a computer science teacher, she is very capable and will have no problems using this software
- Accompanists and music teachers: Often working part time, this group can often be less comfortable with technology.
- Students: Can be irresponsible and immature, and may be confused by a complex interface. More likely to access the website through a mobile device.

As Ms Pearcey is much more experienced with technology than the other users (on average), it is very important that the pages which are viewed by **all** users are easy to use, and have a simple interface (eg. the login page, the index page etc). As I plan to make this software available to schools across the country in the future, it is still important to incorporate this simplicity in my design throughout my project. A method of making this system easy to use involves regularly displaying helpful messages on screen. This can direct confused users in the right direction, by using the `<a>` tag to link to another page which may possibly be more helpful. Furthermore, it allows users to develop an understanding of the software, so they can operate it more efficiently over time.

As my program takes many inputs, in several different forms, validation is key. Validation ensures that all data that is inputted into the system is sensible, and they are all formatted correctly to ensure that the next step of the process works correctly. All validation will be addressed in section [2.4 Data](#). In order to help users with the process of submitting the correct information, specific error messages will be displayed based on the issue. This will help users to locate and fix the problem easily. I also aim to provide helpful messages at some points to prevent these issues from occurring. This includes specifying the format for examinee information clearly, and displaying the password requirements during user registration. I will also use functions to prevent SQL injection, such as the `mysqli_real_escape_string` function, and functions to prevent unexpected HTML outputs, such as `htmlspecialchars`.

Many of the users, especially students, will access this website through a mobile device. Therefore, it is important that my solution is compatible with both desktop and mobile screens. This is why I have decided to use the Bootstrap framework, a very popular framework for building responsive, mobile-first sites. This will become more apparent in development, but Bootstrap makes it easier to create a website that adjusts the sizes of elements based on screen size, making it compatible with a wide range of screen sizes. The choice of using Bootstrap not only allows me to create professional websites that are easy to navigate, but allows me to make this software compatible with any device, mobile or desktop.

One key usability feature involves the design of the ‘schedule’ and ‘publish’ function. When talking to Ms Pearcey in the earlier stages of this project, she introduced me to Practice Pal.

This is a software she uses to schedule music lessons (not exams). When researching this software in section [1.4.2 Practice Pal](#), I came across the simple 'publish' and 'schedule' buttons. These buttons were clearly highlighted, inviting the user to click it. They included an icon, and through coordinating colours across the webpage, this clearly stood out on the screen. Not only was the button styled well, but the existence of a simple button to carry out several tasks shows that the system is well-designed. Inspired by Practice Pal, my software is being designed with the hope of including similar features. The simplicity of Practice Pal revolves around the limited amount of data that needs to be entered. Through efficient database design (see [2.4 Data](#)), I can ensure that Ms Pearcey only has to enter the data required, and is able to click an inviting, highlighted button to 'magically' schedule all exams. Similarly, the publish function will not keep asking for user inputs. By clicking publish, a single function will automate several processes all at once. This design ensures that I can create a 'publish' button that, again, 'magically' does everything for Ms Pearcey. This prevents her from feeling overwhelmed by the system, and instead feeling comfortable with its functionality.

As seen on the first page of this document, the software will be named SoundBro. The name "SoundBro" was thoughtfully selected to represent a sense of harmony and collaboration within music education. The term "Sound" naturally ties into the musical context, while the "Bro" brings a sense of familiarity and friendliness. This name conveys ideas of collaboration in music, but also creates an informal and playful atmosphere in the software. This is done intentionally, as many examinees can often feel very stressed about their upcoming music exams. By creating this informal feeling, the name establishes a connection among users to create a community feel, and make users feel more comfortable with the software. It also appeals to students and teachers alike, as it strikes a balance between professionalism and a relaxed, inviting tone. Using canva.com, I have designed a logo for this software:

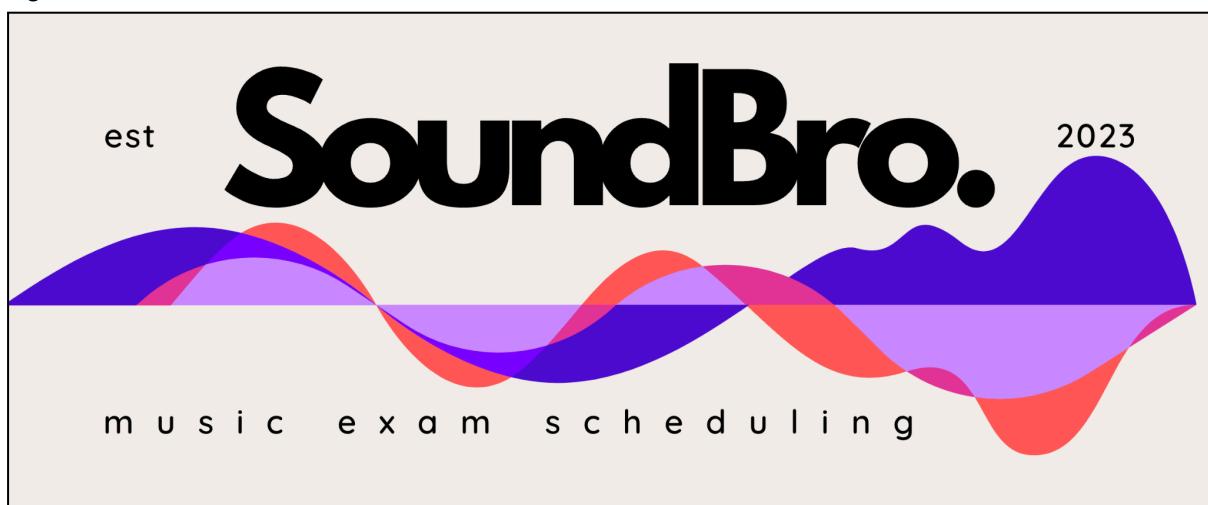


Figure 2.3a: Main SoundBro logo



Figure 2.3b: Secondary SoundBro logo | Figure 2.3c: 'favicon' for website.

This logo reinforces the idea of simplicity and informality in the system. The main logo illustrates an abstract image of a soundwave, along with the subtitle ‘music exam scheduling’ to reinforce the musical context. The font used is distinctively different from usual fonts such as Arial, which will help it to stand out on the page. I aim to use the main logo in the homepage, but the text of **SoundBro** will be displayed on every page, on the navigation bar. Figure 2.3c shows the favicon I have designed for the website. This will be visible in the tab for this website, as shown in the example below.

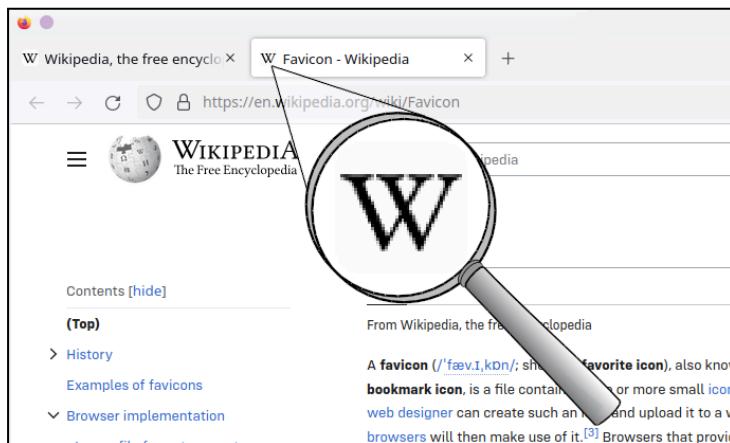


Figure 2.3d: Wikipedia favicon

The combination of name and logo, along with colour coordination across the webpage will cultivate a sense of comfort and familiarity, allowing the user to develop a connection with the website. This will help them to use and navigate the website, ultimately resulting in a positive and engaging user experience.

### 2.3.1 Web design

As mentioned, the user must feel comfortable to use and navigate the website. This is especially important for the initial pages of the website, such as the home page and login page. This is because all users will interact with these pages, including users who will rarely use the website. Although Ms Pearcey will become familiar with this software as she repeatedly uses it, many users will log in two or three times each year, simply to check their exam time and nothing else. These users will not be familiar with the software, so it is crucial that these pages look professional and inviting. Most importantly, they should be simple, abstracting anything that is not necessary in the page. I decided to design the login page

here, as it will be used by everyone, so needs to be carefully thought out. Version 1 of my login page is displayed below.

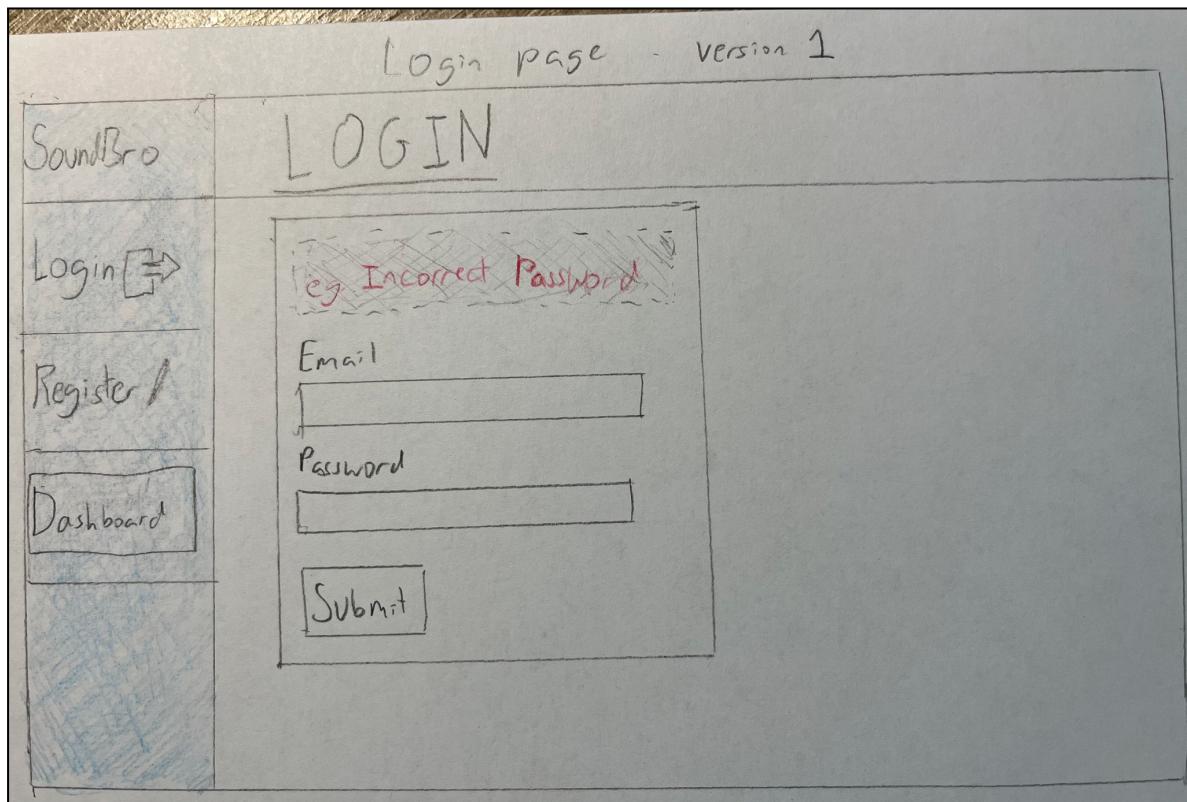


Figure 2.3.1a: Login page design - version 1

This was the first iteration of my login page. I tried to keep the display simple, using blue, red and white colours only. I included a navigation bar to help users navigate through the website, placing it horizontally to free up space for the main content. I also felt that the horizontal navigation bar was more modern and natural. However, I felt that it was not perfect, and did not achieve the simplicity I was striving for. Therefore, I sent this image to my student stakeholders, Cyrus and Darius. Cyrus and Darius are both in year 13 and used to accessing many websites for various purposes. I sent this to both stakeholders, but listened more to the feedback from Cyrus. This is because Cyrus has a keen interest in graphic design, so is quite knowledgeable about this aspect of the project.

### *Feedback from Darius*

"It looks okay, but a bit basic. I don't like those lines at the top, and capitalising the whole word login looks unprofessional. That box in the middle looks pretty good. But the incorrect password bit doesn't stand out enough. I don't think blue is a great colour, maybe something like orange would be more fun and it would stand out more."

### *Feedback from Cyrus*

"The navigation bar doesn't have enough elements to be horizontal. The space wasted at the bottom of the navigation bar makes it look unfinished I think. Probably best to put it at the top. I like the little icons for login and register. The login box is in an odd position, you need

to align it to the centre, or to the left. I was thinking you could make the box wider and get rid of that empty space on the sides. Overall the page looks decent but a bit bland. Whatever you decide I would try to be consistent throughout the website."

It was refreshing to have another person give feedback on this design. I was so accustomed to what I had done, which prevented me from seeing some obvious mistakes. I immediately agreed with Cyrus' suggestion of moving the navigation bar to the top. I decided to keep the blue colour, going against Darius' comment to make it orange. I feel that orange could be too bright, but blue finds a good balance between highlighting something without distracting from the rest of the page. I also centralised the login container and the submit button, and highlighted this button in blue. I sent this back to each stakeholder, who suggested some minor adjustments. When I gave them my third version, they seemed happy with the layout and structure of the page.

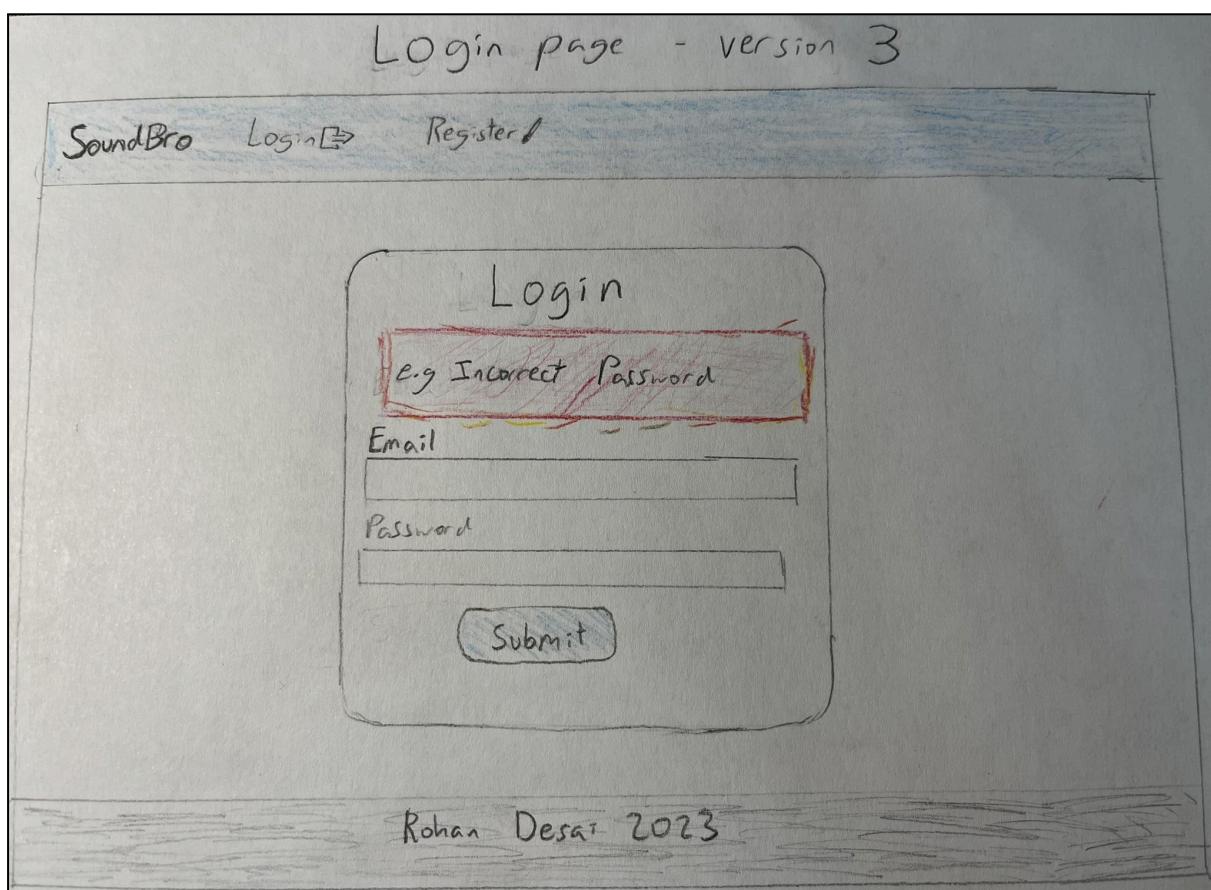


Figure 2.3.1b: Login page design - version 3

You can see that, compared to the first version, some significant adjustments have been made. Firstly, I have decided to modify the navigation bar. As Cyrus suggested, it is now at the top of the page, which makes the whole page look simpler. I have removed the borders between each link in the navigation bar, and made the 'SoundBro' text bolder to make it stand out more. I have also removed the link to the dashboard. This is because, if someone has not logged in yet, they cannot access the dashboard yet. If they were to go onto the dashboard, they would be redirected back to the login page. Therefore, to maintain simplicity and avoid confusion, the dashboard link will only be available on the navigation bar if the

user has logged in. When logged in, the navigation bar will again only display the links that are possible, eg. ‘login’ and ‘register’ will disappear, and ‘logout’ and ‘dashboard’ will replace them. Using icons provided by Bootstrap 5, I will find a matching icon for each link in the navigation bar. I think that the icons make each link more recognisable, and clarifies the purpose of the link. At a glance, users will be able to navigate to where they want to go as a result of these intuitive icons. I plan to use icons for all major buttons and links for this reason, including the schedule button and publish button.

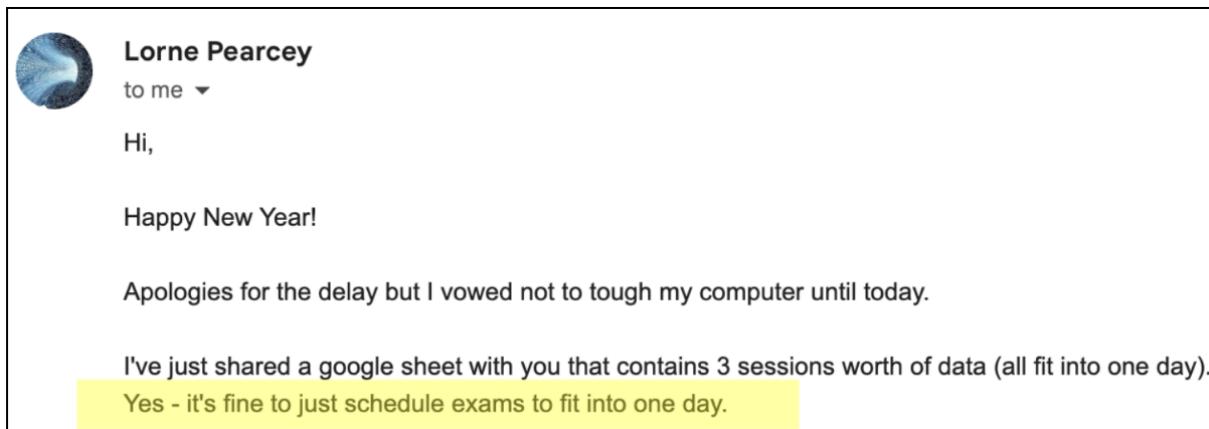
The blue colour is the primary colour for this website. This colour will be used to highlight key buttons throughout the entire website. The submit button is now highlighted in blue, encouraging users to click it. The login container and submit button have been centred to keep the page professional. The rounded corners of the login container were suggested by Cyrus when I gave them my second version. I believe that this change makes it look more modern and simplifies the design. It also highlights the main purpose of the page, encouraging users to enter their login credentials. Cyrus suggested making the box wider, removing empty space on the sides. However, this would make the website less mobile-friendly, as containers would be compressed significantly to fit into the mobile screen size. Therefore, I have not decided to include this feature, as it would make the website less accessible by a mobile device.

To highlight errors more clearly, the entire box will have a red background colour, not just the text. The red colour indicates that there is a problem, and by making the entire box red the user is immediately drawn to this problem. I have also added a footer, displaying my name and the year. This indicates the end of the page, clarifying to users that they do not have to do anything else.

I believe that the features described significantly enhance the user’s experience and allow them to interact with the page much more easily. I will be consistent with my style throughout pages, which means coordinating colours, fonts, and other styling elements to make each page feel familiar.

## 2.4 Data and Validation

**Note: After facing significant difficulties with storing accompanist availability, the software will now only schedule exams for one day.** When trying to create a database that stores availability I faced many issues. Some of the most challenging issues were as a result of the unknown end date. A potential solution involved accompanists recording availability in 'blocks' (eg. 9am-1pm, then 10am-12pm on day 2), but this led to many-to-many relationships between the 'Blocks' table and Examinees, as each examinee could be scheduled into any one of their accompanist's blocks. As I tried to create more tables to remove many-to-many relationships, the database started to become quite large and it was apparent that this issue was causing my project to become exceedingly difficult, and outside the scope of this project. Therefore, I consulted my client, Ms Pearcey, and asked if I could schedule exams on one day only (this was something she had already suggested in previous discussions).



The screenshot shows an email interface. On the left is a circular profile picture of a person with blue hair. To the right of the picture, the name "Lorne Pearcey" is displayed, followed by a small dropdown arrow. Below the name is the recipient "to me ▾". The main body of the email starts with "Hi," followed by a blank line. Then "Happy New Year!" is written. Another blank line follows, then "Apologies for the delay but I vowed not to touch my computer until today." A yellow callout box appears over the text "I've just shared a google sheet with you that contains 3 sessions worth of data (all fit into one day)." The callout box contains the text "Yes - it's fine to just schedule exams to fit into one day." The entire email is enclosed in a light gray border.

Lorne Pearcey  
to me ▾

Hi,

Happy New Year!

Apologies for the delay but I vowed not to touch my computer until today.

I've just shared a google sheet with you that contains 3 sessions worth of data (all fit into one day).  
Yes - it's fine to just schedule exams to fit into one day.

Figure 2.4a: Ms Pearcey confirming I can schedule exams to fit into one day (highlighted).

I emailed her to ask if it was possible to remove this feature, explaining some issues I had faced ( I also asked for some test data, please ignore this for now - it will be addressed in section [2.6 Iterative test data](#) and [2.7 Post-development test data](#)). She emailed back to say that it was fine, so together we agreed to remove this feature. This document is in chronological order of my progress, so from now on my project will reflect this update.

This project involves the collection of lots of data, most of which will be stored in databases. The majority of this data must be validated, and all of it must be stored efficiently so it is easy to retrieve later on. Data validation ensures accurate, consistent, and compliant data entry so the information stored is more reliable and the system can be trusted. It prevents 'garbage in, garbage out' and therefore is a crucial aspect of this solution.

### 2.4.1 Database design

Databases will store the majority of my data, and will be used throughout all parts of the solution for several purposes. The process of designing my relational database involves finding a way to structure the data so that it is stored in the most efficient way possible, with all tables normalised to third normal form (3NF), and no many-to-many relationships. I

believe that my designed database will achieve this level of normalisation, which reduces data redundancy and simplifies maintenance of the database.

During the analysis stage, I highlighted a limitation of my project in [1.5.5 Limitations](#). The issue was that one person can be both an accompanist and a music teacher, so in this situation this person would need 2 different logins. I have fixed this issue by adjusting how each user is handled by the program, and how their data is stored. The key change which has allowed for this issue to be solved is in my database design. Originally, I designed a relational database with one entity for accompanists and one for music teachers. Having the same details across 2 entities would be redundant data, which can become very difficult to manage and often leads to inconsistent data. Upon further discussion with Ms Pearcey, I learned that many teachers accompany their students, so this situation is not so uncommon that I can ignore it. When I brought up this limitation, she did mention that it would be nice for one person to have just one login. Therefore, I decided to try again and change my approach by creating one entity called 'Supervisors' and one called 'ExamAccomps'. The first table contains basic data for music teachers **and** accompanists, so there is no repeated data. There is no distinction between teachers and accompanists in this table, which allows any teacher to accompany their student. The second table, ExamAccomps, contains availability data for all accompanists in the current exam cycle. Examinees will have one music teacher, and one accompanist, so in the 'Examinees' table there is a link to Supervisors (for music teachers) and a link to ExamAccomps (for accompanists). Therefore, the system does not view people separately as 'accompanists' and 'music teachers'. Instead, it sees 'supervisors' and 'accompanists in the current cycle', where one person can fall into both categories without data redundancy. This section will show that my solution involves no many-to-many relationships and can be normalised to the third normal form. By adjusting my method, this limitation has been fixed, and each person will only need one login.

There is no mention of Ms Pearcey in this database design. This is because she is the only organiser, therefore I will develop code to recognise her email during login so she can access all features of this program as an administrator. As Ms Pearcey is a music teacher and accompanist herself, her data will be stored in the 'Supervisors' entity, allowing her to become part of the system as a supervisor, and an administrator.

### *Entity Relationship Diagram*

I have aimed to design the relational database as efficiently as possible. Initial designs involved storing all users in one database, storing student and examinee information together, and by identifying and correcting my mistakes I have created the entity relationship diagram below.

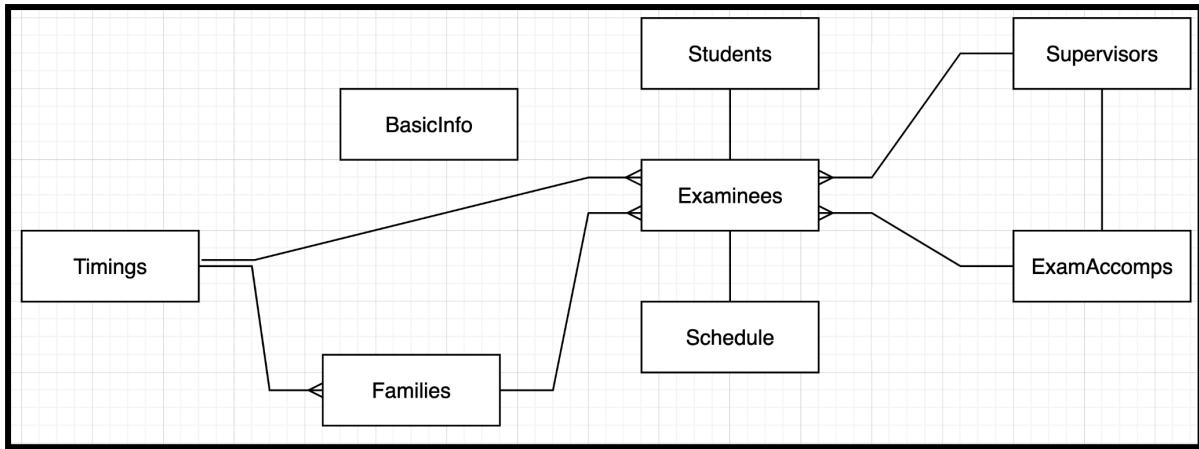


Figure 2.4.1a: Entity relationship diagram (ERD) of database.

The database will be named after the product: SoundBroDB. This diagram clearly illustrates the different relationships between each entity in the database. BasicInfo does not have any relationships, as one record in that table relates to the entire exam cycle. BasicInfo could have been instead stored as a text file, but I believe that this is a much simpler way of storing this data, and makes the system more suitable for future developments. This table could be used to reference old schedules, and it is easy to alter its structure to add any fields if needed. It also ensures that all key data is stored in one database, making all data centralised simplifies the data structure and prevents additional complexity in the system. This diagram can be developed further to show each field, which is shown below.

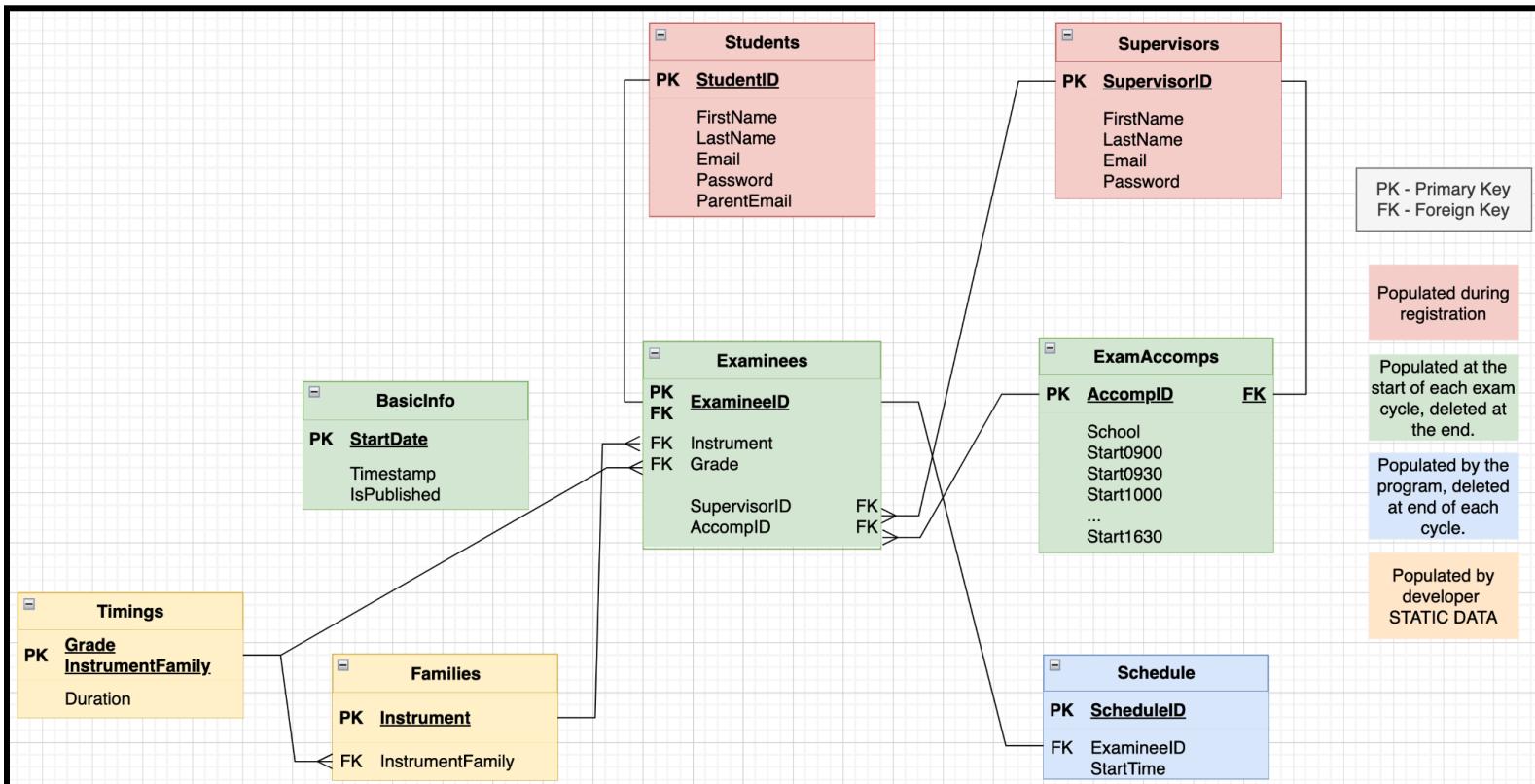


Figure 2.4.1b: Detailed ERD, containing field names and other information.

The first tables to be populated will be Timings and Families. These contain static data related to the TCL regulations, so will be created during development. Then, Students and Supervisors will be populated as the system is released and users start to register. They will also be filled if Ms Pearcey uses CSV files to register users in bulk. When Ms Pearcey starts a new exam cycle, BasicInfo will be created. One record in this database will contain initial information about this schedule. ExamAccomps and Examinees will then be filled through user inputs and CSV file inputs. All of these tables are validated as they are populated. Finally, Schedule is created when the ‘schedule’ function is run. When editing a schedule, the StartTime of all records is updated, and then it is published by setting BasicInfo.IsPublished to True. At the end of the cycle, BasicInfo, Examinees, ExamAccomps, and Schedule are deleted.

Using phpMyAdmin and MySQL, this database will be created. It will be linked to my program through PHP code and its MySQLi extension. This database allows me to:

- Store data for each user
- Store TCL information related to exam timings and instruments
- Store availability for accompanists
- Store examinee information
- Create and store the final schedule

Below I have explained and justified why the database has been designed this way. Please note that the data types given below are based on phpMyAdmin’s allowed data types. VARCHAR(191) is the same as a string. 191 refers to the maximum number of characters, and MySQL databases work best with this number for compatibility and performance reasons. ‘Not null’ indicates that the field is required, so all fields with this attribute will have the additional validation of a presence check.

### FAMILIES

Field name	Data type	Validation	Attributes	Notes
Instrument	VARCHAR(191)	None	<b>PRIMARY KEY, Not null</b>	
InstrumentFamily	VARCHAR(191)	None	Not null	FK for Timings

Figure 2.4.1c: FAMILIES table and validation

### TIMINGS

Field name	Data type	Validation	Attributes	Notes
Grade	INT	None	<b>PRIMARY KEY, Not null</b>	Composite primary key
InstrumentFamily	VARCHAR(191)	None	<b>PRIMARY KEY, Not null</b>	Composite primary key
Duration	INT	None	Not null	

Figure 2.4.1d: TIMINGS table and validation

As the detailed ERD shows, Timings and Families are static databases, and are populated by the developer. Therefore, entries into these tables do not need to be validated (as they will be filled by me). Foreign keys which reference these tables will need to be validated, but the entries into these tables do not. The purpose of these tables is to get the duration of an exam based on the instrument family and grade, so that the scheduled duration of each exam follows TCL regulations. These rules can be found in figure 1.1.2b, which has been copied below.

LEVEL	EXAM TIMINGS (MINUTES)				
	Piano	Drum kit and percussion	Harp Organ Rock & Pop exams Electronic keyboard	Brass Guitar Strings Singing	Woodwind
GRADE EXAMS					
Initial	10	13	13	11	
Grade 1	11	15	13	13	
Grade 2	11	15	15	13	
Grade 3	12	16	15	13	
Grade 4	16	21	20	18	
Grade 5	16	21	20	18	
Grade 6	22	27	25	23	
Grade 7	22	27	25	23	
Grade 8	27	32	30	28	

(repeated) Figure 1.1.2b: Duration of each grade exam

If each instrument family and grade has a specific duration, then each instrument and grade must have a specific duration. For this reason, I originally combined the 2 tables, so that instrument and grade from Examinees directly referenced a composite primary key of these fields. This was a simpler structure, however I later realised this was inefficient, and I did not have the ability to access an instrument's family. It is important that each examinee can be linked to their instrument's family, as this is used to group exams when scheduling. I considered including the field InstrumentFamily in Examinees. However, this would be inefficient as Ms Pearcey would have to enter more data than necessary, and would have to enter the exact name of the family. This would introduce the possibility of errors. Furthermore, the Examinees table would no longer be in 3NF. As each InstrumentFamily is entirely dependent on Instrument, this is a transitive dependency, and means that a field in the table is not entirely dependent on the primary key. Therefore, I created Families to improve efficiency and maintain 3NF in my database. As each family depends entirely on the instrument, and there are no repeated attributes, this table is in 3NF. Similarly, in the Timings table, there are no repeated attributes. To maintain simplicity and avoid creating unnecessary IDs, Grade and InstrumentFamily form a composite primary key. As figure 1.1.2b shows, the duration of the exam depends on both the grade **and** the instrument family, so this table is in 2NF. As there are no other fields, Duration cannot be dependent on any other fields, so this table is in 3NF. The choice of using InstrumentFamily as part of the

primary key (not Instrument) also ensures that the table is as efficient as possible, and records do not have to be repeated multiple times. If Instrument and Grade formed a composite primary key, the record for trumpet grade 1 would be identical to trombone grade 1. Now, there will just be a single record: brass grade 1.

I faced considerable difficulty with determining the nature of relationships between Timings, Families and Examinees. The main issue was managing a composite primary key where its foreign key is split between 2 tables. The logical approach I took resulted in the ERD shown above. One examinee will play one instrument. However, one instrument can be played by many examinees. Therefore, Families to Examinees is one to many. One examinee will play one instrument, and one instrument relates to one instrument family. Therefore, each examinee will have a specific instrument family. Each examinee also plays at one grade. Therefore, one examinee will have one combination of grade and instrument family, so each examinee can be linked to a single duration. However, each instrument family can have many instruments, and each grade can have many examinees. Therefore, Families to Timings is many to one, and Examinees to Timings is many to one. To conclude, these tables are in 3NF and have no many to many relationships. They contain all required data in the most efficient solution, and all data can be accessed easily using SQL.

All information in both tables will be in lowercase, which makes it easier to compare to their foreign keys in the Examinees table. This simplifies the validation process for the CSV of examinee information. The names of Instrument and InstrumentFamily will be exactly as given in [1.1.2 TCL Regulations](#).

### STUDENTS

Field name	Data type	Validation	Attributes	Notes
StudentID	INT	None	PRIMARY KEY, Auto-increment, Not null	
FirstName	VARCHAR (191)	Length check: 2-50 characters Format check: no numbers or special characters (except hyphens)	Not null	
LastName	VARCHAR (191)	Same as FirstName	Not null	
Email	VARCHAR (191)	Validate against RFC 822 Check that it is not already in Students or Supervisors.	Not null, UNIQUE	
Password	VARCHAR (191)	Length check: at least 8 characters Format check: at least one uppercase letter, one lowercase letter and one number <i>Verification: entering password twice during registration</i>	Not null	Stores hashed password

ParentEmail	VARCHAR (191)	Validate against RFC 822	Not null	
-------------	---------------	--------------------------	----------	--

Figure 2.4.1e: STUDENTS table and validation

### SUPERVISORS

Field name	Data type	Validation	Attributes	Notes
SupervisorID	INT	<b>None</b>	<b>PRIMARY KEY, Auto-increment, Not null</b>	
FirstName	VARCHAR (191)	Length check: 2-50 characters Format check: no numbers or special characters (except hyphens)	Not null	
LastName	VARCHAR (191)	Same as FirstName	Not null	
Email	VARCHAR (191)	Validate against RFC 822 Check that it is not already in Students or Supervisors.	Not null, UNIQUE	
Password	VARCHAR (191)	Length check: at least 8 characters Format check: at least one uppercase letter, one lowercase letter and one number <i>Verification: entering password twice during registration</i>	Not null	Stores hashed password

Figure 2.4.1f: SUPERVISORS table and validation

Note: The Supervisors table stores data for accompanists and music teachers. As all fields are required, they will all have the additional validation of a ‘presence’ check. These tables have been created to store initial information about students and supervisors, and are populated during registration. The tables contain as much information about the students/supervisors that will **not** change with each exam cycle. All of this data will be used each time exams are scheduled, but they will not change from cycle to cycle. This means that the data can be stored once, in a separate table, and prevents Ms Pearcey from having to collect it every time. This is the important distinction between Students and Examinees: the data contained in Students is static, whereas the data in Examinees is dynamic. The Examinees table can access all of this static information easily, and all Ms Pearcey has to input is the student’s email address. The same is true for Supervisors and ExamAccomps: data in Supervisors will not change, and data in ExamAccomps will change with each cycle, but all data can be accessed easily, and the same data does not have to be repeatedly entered. This is possible by simply dividing the student / supervisor data into 2 tables each.

The reason why the Email field is unique is so that it can uniquely identify each record in the field, and prevents users from registering multiple times by accident. I considered making this field the primary key, but emails can sometimes be changed, so instead chose to make StudentID and SupervisorID. Unique emails also allow an email to link tables together,

instead of an ID which is difficult to remember. This is especially useful when the CSV file of examinee information is entered. When it comes to validating emails, I am already aware that `<input type="email">` validates emails automatically. However, I wanted to be more rigorous, and after some research found RFC 822. RFC 822 is a specification that defines the syntax for email addresses, so I would like to validate emails against these regulations. I will likely use the PHP function `filter_var` along with the filter `FILTER_VALIDATE_EMAIL`, as this function already validates email addresses against these rules, with a few minor exceptions. As students cannot be supervisors, no one should be registered twice in the system. Therefore, the program will ensure that the entered email does not already exist in Supervisors or Students. All students will have to enter their parent's email address because parents will also be emailed when the schedule is published. First and last names are validated sensibly to deter misbehaviour from students. StudentID and SupervisorID are automatically created when creating a new record in the table, so do not need any validation.

The password keeps a level of security, ensuring each account is only accessible to the desired person. To increase this security, passwords are validated using standard format and length checks, and will be hashed before being stored in the table. To prevent accidental typos from changing the password, it will have to be entered twice during registration. This form of verification only stores the password if the same data is entered twice, which reduces the likelihood of errors going undetected. Although the data being sent to each account is not particularly sensitive, it provides a level of security that stops immature students from tampering with the system. As many people use the same passwords for different systems, developing a database which stores everyone's passwords can pose serious security concerns. Therefore, the one-way hashing process prevents passwords from being leaked, as their hash cannot be reverse engineered. This enhances the safety of users, and ensures that no one can view other people's passwords, not even the administrator.

Originally, the student's music teacher was stored in Students. This led to a dilemma when I realised that one student can play multiple instruments and have multiple teachers. I chose to make the assumption that one examinee will only sit one exam in a cycle (according to Ms Pearcey, it is very rare for examinees to sit 2 exams in one cycle). Therefore, the music teacher relationship will be made in Examinees, which allows the student to have multiple teachers.

These 2 tables can be populated by CSV files. In this situation, there will be additional validation to check that the file is in fact a CSV file, and checking that it has the correct number of rows. Once the file has been confirmed to be in the correct format, each record will be validated with the same rules as specified above.

In the Students table, all fields are atomic. Every field depends on the StudentID, and there are no transitive dependencies. Therefore, this table is in 3NF. The Supervisors table also has no repeated attributes, and no transitive dependencies or other inefficiencies. This table is also in 3NF.

### BASICINFO

Field name	Data type	Validation	Attributes	Notes
StartDate	DATE	<b>Format check: YYYY-MM-DD Range check: date must be in future</b>	<b>PRIMARY KEY, not null</b>	
Timestamp	TIMESTAMP	None	Not null	Automatically set to current time
IsPublished	BOOLEAN	None	Not null	Automatically set to False

Figure 2.4.1g: BASICINFO table and validation

This table is the only table in the database with no relationships. This is because one record in BasicInfo relates to the entire Schedule table. However, each field is crucial to the structure of the development and the success of the solution. It is possible that this data could be stored in a text file. However, this would introduce another data storage system, making the solution more complicated for little reason. Some may argue that it should simply be stored as a session variable, but this would force Ms Pearcey to go through the entire scheduling process in one sitting. The key issue that many will have with this entity is that it will only ever contain one record. Therefore, it may seem unnecessary to have this large structure to only hold 3 pieces of data. However, this has been designed intentionally. Although it will not be completed within the timeframe of this project, I aim to develop a more flexible solution. This future iteration will save old schedules and allow them to be loaded, preventing Ms Pearcey from having to go through each intricate step of the scheduling process. In this iteration, BasicInfo will store this information about each schedule that is created, so the database will have one extra record added for each exam cycle. This method also allows me to add another field very easily, so if a particular aspect of the project needs to be edited or improved, there is flexibility in the database to do so.

The StartDate field is the primary key for this table. It will tell the system the date of the first exam, which is a very important date to store. This date will be displayed when asking accompanists for availability (ie. “Are you available on DD/MM/YYYY ?”), and will be displayed on all dashboards, emails and the final PDF when published. This date could be stored in another table, eg. included inside the StartTime in Schedule, however these tables will be created too late in the scheduling process. As the start date will be used in many different situations, it is important to store it clearly in this table. It will be validated to ensure that the date is in the future, and if not it will return an error message. The Timestamp field is very useful for the organiser, especially if they need to keep track of schedules created. This will have no user input, the date and time will automatically be recorded when the user clicks ‘submit’ to enter the start date. According to MySQL documentation, the following data types are stored as this:

- DATE: YYYY-MM-DD
- TIMESTAMP: YYYY-MM-DD hh:mm:ss

Therefore, the Timestamp will be saved in this format by the program. The StartDate, however, has user input and therefore validation needs to be more secure. After researching how dates can be inputted, I found the HTML tag <input type="date">. This prevents

users from entering invalid dates (such as months exceeding 12), and users cannot enter any characters other than integers. This ensures that the StartDate will contain a real date, and nothing else. However, the StartDate cannot be in the past or on the current day.

Therefore, I will additionally validate the StartDate by ensuring it is in the future. If a past date was accepted and stored, it could cause significant errors with email scheduling, and lead to misunderstandings between all users.

The field IsPublished is initially set to False by the program. When the organiser clicks 'publish', this field is set to True. This change will enable the schedule to become visible to all users. In this table, there are no repeated attributes. The fields TimeStamp and IsPublished depend entirely on the primary key StartDate, and do not depend on any other fields in the database. This table is in 3NF.

### EXAMACCOMPS

Field name	Data type	Validation	Attributes	Notes
AccompID	INT	None	<b>PRIMARY KEY, Not null</b>	<b>FK for Supervisors</b>
School	BOOLEAN	If True, check that no record already has this field as True.	Not null	
Start0900	BOOLEAN	None	Not null	
Start0930	BOOLEAN	None	Not null	
Start1000	BOOLEAN	None	Not null	
Start1030	BOOLEAN	None	Not null	
Start1100	BOOLEAN	None	Not null	
Start1130	BOOLEAN	None	Not null	
Start1200	BOOLEAN	None	Not null	
Start1230	BOOLEAN	None	Not null	
Start1300	BOOLEAN	None	Not null	
Start1330	BOOLEAN	None	Not null	
Start1400	BOOLEAN	None	Not null	
Start1430	BOOLEAN	None	Not null	
Start1500	BOOLEAN	None	Not null	
Start1530	BOOLEAN	None	Not null	
Start1600	BOOLEAN	None	Not null	
Start1630	BOOLEAN	None	Not null	

Figure 2.4.1h: EXAMACCOMPS table and validation

## EXAMINEES

Field name	Data type	Validation	Attributes	Notes
ExamineeID	INT	Check that it exists in Students*	PRIMARY KEY, Not null	FK for Students
Instrument	VARCHAR (191)	Check that it exists in Families	Not null	FK for Families
Grade	INT	Range check: must be from 1 to 8	Not null	FK for Timings
SupervisorID	INT	Check that it exists in Supervisors*	Not null	FK for Supervisors
AccompID	INT	Check that it exists in ExamAccomps*		FK for ExamAccomps

Figure 2.4.1i: EXAMINEES table and validation

\*The user's email is checked, not their ID. This is explained later on.

These 2 tables, as well as BasicInfo, will be created during the early stages of each exam cycle, and deleted at the end. ExamAccomps will store availability for each accompanist, along with the boolean field School. Examinees will store examinee information and, as the ERD shows, links all parts of the database together. This data is crucial, as it provides essential information to be used for scheduling exams.

In ExamAccomps, the table will be populated through an interface provided on the website. The accompanists who have been selected by Ms Pearcey will be able to log on and select availability. By logging on, their SupervisorID will be recorded. The page will display the day split up into 30 min slots (from 9am to 5pm), so that the accompanist will be able to click each slot to indicate availability. There will also be a tickbox asking 'Are you the school accompanist?'. If this box is selected, an algorithm will check if another accompanist has already said they are the school accompanist. If another accompanist has done this, an error message will be displayed. If not, the result will be processed. Alternatively, Ms Pearcey can select an accompanist and arrive at the same page to fill out this form for them. In either situation, the fields which record availability (Start0900 etc.) and the School field will only ever be 1 or 0. This is because the interface only allows the inputs to be 1 or 0 by pressing the timeslot or not. Therefore, the fields School and Start0900,Start0930 ..., Start1630 **do not** need to be validated.

Originally, I attempted to design a table that records availability over multiple days. I tried to take into account that an accompanist can be available in multiple blocks, so they may be available from 9am to 11am, then 4-5pm. This prevents me from including 'StartTime' and 'EndTime' for each accompanist, as it limits them to only be available for one of these blocks. However, an examinee can fit into any one of their accompanist's blocks, and each block can have many examinees. This is a many-to-many relationship and therefore I tried to create an additional table displaying all possible combinations of examinees and blocks, Examinee\_Blocks. While this solution worked, it was very complex and made every other aspect of the scheduling process more complicated. As I tried to develop the main scheduling algorithm, I faced several issues, especially with the scheduling of examinees

without accompanists. I thought about simplifying the scheduling process, but this caused me to re-evaluate my project's direction. My main goal is to schedule exams as efficiently as possible, so I should make this my top priority. This is what pushed me to initiate discussion with Ms Pearcey, asking to schedule exams on 1 day only. In order to focus on the key features of the project, exam scheduling will only be supported for 1 day only.

Now that my exams were in a fixed timeframe, I was able to create the current solution. Once again, I couldn't simply have a StartTime and EndTime, forcing availability to be in one block. Therefore, I made the compromise of dividing availability into 30 minute boolean fields, so accompanists have the ability to declare availability more accurately. The fields are boolean as the accompanist's can only be available or unavailable at a certain time. The limitation of this is that they can only be available in 30 minute slots. However, this issue is not detrimental to the solution, as most accompanists are music teachers, a group who are used to the education system and fixed blocks of lessons. After viewing real-life examples of accompanist availability provided by my client, my thought process was validated; accompanist availability was already being recorded in 30 min slots. I have talked in more detail about the test data in [2.6 Iterative test data](#) and [2.7 Post-development test data](#). As each field in ExamAccomps is entirely dependent on the AccomplID (primary key), there are no transitive dependencies. There are no repeated attributes or other issues in this table, it is in 3NF. This ensures that the database has no redundancy, and minimises complexity. The simpler a database design is, the easier it will be to make queries to it during development.

It is important to note that AccomplID is **not** a new ID that is created. AccomplID is equal to SupervisorID. The same is true in the Examinees table: ExamineeID is equal to StudentID. While this may appear to be confusing, I find that it is simpler than using TableName.FieldName . When an accompanist logs on to enter availability, their SupervisorID will be stored. When they submit this availability, their SupervisorID will be used to make an entry into ExamAccomps, and AccomplID will take the value of the SupervisorID (which is why AccomplID is not validated). The difference between the names doesn't change that the 2 tables can be joined together, as a user with SupervisorID=2 will have AccomplID=2. The different names also highlight that each ID represents a different group.

- StudentID: for all registered students
- ExamineeID: for all examinees in the current exam cycle, this is a selection of records from the Students table
- SupervisorID: for all registered supervisors (music teachers and accompanists)
- AccomplID: for all accompanists in the current cycle, this is a selection of records from the Supervisors table.

The main benefit of this naming system can be seen when observing the fields in Examinees. Every examinee has one music teacher. Every examinee has (a maximum of) one accompanist. One music teacher can have many examinees, and one accompanist can have many examinees. However, one music teacher can also be an accompanist. This thought process is why music teachers and accompanists are stored in a single table: Supervisors. ExamAccomps, on the other hand, only stores data for accompanists. Therefore, each examinee's accompanist **must** be in ExamAccomps (if they have an

accompanist). And each examinee's music teacher **must** be in Supervisors. Therefore, each examinee will have a SupervisorID to link to their music teacher, and AccomplID to link to their accompanist. This supports the possibility of the same person taking on both roles! The different names makes it much easier to understand the fields in Examinees. SupervisorID and AccomplID may be the same for each accompanist, but every music teacher will have a SupervisorID and every accompanist will have an AccomplID.

As Ms Pearcey is handling this data, there is little chance of errors when pairing students to their teachers. Therefore, my design has no way to distinguish between music teachers and accompanists in the Supervisors table. This means that there is no way to check if Ms Pearcey accidentally says that an accompanist is someone's music teacher. However, I am confident that she will not make this error, as she has been scheduling exams for over 20 years. To summarise:

- AccomplID is a copy of SupervisorID
- ExamineeID is a copy of StudentID
- In Examinees, the foreign key 'SupervisorID' refers to music teachers
- In Examinees, the foreign key 'AccomplID' refers to accompanists

The purpose of storing Instrument and Grade is so that a matching duration can be found. This duration is used to increment the time when this examinee is scheduled. Each examinee is linked with their music teacher so that their teacher is emailed about when this examinee (and any other examinees they teach) has their exam. Finally, each examinee is linked to their accompanist, or specifies that they don't need an accompanist. This is crucial as it determines which examinees should be scheduled in order to meet accompanist availability and schedule exams efficiently.

ExamAccomps doesn't need any validation (except for the School field) because the interface provided will limit the types of inputs that can be entered. However, the Examinees table is entirely populated by a CSV file. This file must be validated in several ways, and there are many steps to get from that file to a full table.

#### *The CSV file for examinee information*

At this stage in the process, Timings, Families, Students, Supervisors, BasicInfo, and ExamAccomps are full. The screen will display a button, 'choose file', that allows Ms Pearcey to submit a CSV file from her laptop. This will be possible using the HTML tag `<input type="file">`. Using `accept=".csv"`, I can encourage her to submit a CSV file. However, to be secure, I will confirm that it is a CSV file using PHP code, after she has submitted the file. I will then check that the number of fields are correct, and then check the data types are correct. I cannot expect Ms Pearcey to know each user's StudentID or SupervisorID, but as everyone's emails are also a unique index, this will be used to uniquely identify each user. The CSV file must be in the exact format as outlined below for each examinee:

Field Name	Examinee email	Instrument	Grade	Music teacher	Accompanist email
------------	----------------	------------	-------	---------------	-------------------

	address			email address	address OR "school" OR "none"
Data type	String	String	Integer	String	String

Figure 2.4.1j: format of CSV file for examinees

Here is a valid example of part of the CSV file:

```
"exampleexaminee@gmail.com","classical guitar",3,"exampleteacher@gmail.com","none"
"anotherexaminee@gmail.com","trombone",6,"ateacher@hotmail.com","accompanist@gmail.com"
"thirdexaminee@gmail.com","violin",8,"musicteacher@hotmail.com","school"
"lastexaminee@gmail.com","flute",5,"teacher@gmail.com","teacher@gmail.com"
```

Figure 2.4.1k: Example CSV file for examinee information (In the final row, "teacher@gmail.com" must be in ExamAccomps as well as Supervisors)

Once the CSV file has been submitted, the following steps will be taken to validate it:

- Is this a CSV file?
- Does each line have 5 fields?
- For each row, does each field have the correct data type?

The CSV file will be converted into an array in PHP. For each row, the algorithm will carry out validation checks on each field and make it suitable for entry into the database. As every field in the Examinees table is a foreign key, the algorithm must check that the field actually exists in the table it is referencing.

For the examinee email address, we need to check that the email exists in the Students table. This is done with the following SQL query: `SELECT StudentID FROM Students WHERE Email = $email`. \$email represents the email address given. If the query returns nothing, the email address is either incorrect, or the student is not registered. An error will be returned immediately displaying this message. As Students.Email is unique, it will only ever return 1 result or no result. If the query returns a matching StudentID, the email address in the array will be replaced by this StudentID. This exact StudentID will later take the name ExamineeID.

For the instrument, we need to check that the given string exists in the Families table. To make the system more robust, I will make it case-insensitive by converting the string to lowercase. Then, the following query will be executed: `SELECT * FROM Families WHERE Instrument = $instrument`. If nothing is returned, an error message is displayed to show that the instrument name has not been found. If something is returned (ie. number of rows >0), we store the lowercase instrument name in place of the inputted text, and continue.

For the grade, we could check that this field exists in the Timings table. However, a grade is always an integer from 1 to 8. As it has already passed checks to confirm it is an integer, we simply check if this number is between 1 and 8. This is simpler than executing another SQL

query. If the number is not between 1 and 8, an error message is returned to say the grade is not correct. If it is, we continue.

For the music teacher's email address, we need to check that the email exists in Supervisors. This process is very similar to the validation for 'Students'. We execute the query: SELECT SupervisorID FROM Supervisors WHERE Email = \$email. If nothing is returned, there will be an error message explaining that this email address is either incorrect or the music teacher is not registered. If a matching SupervisorID is found, the SupervisorID will replace the email in the array. This SupervisorID will later take the name ExamineeID.

For the accompanist's email address, the validation is slightly different. If the field == "none", this particular examinee has no accompanist. In this case, we replace the field with `null` (ie. `$array[i][j] = null;`, where i and j are indexes in the array). This value will later be recognised by the database as a null value, to show that the examinee has no accompanist. If the field == "school", we need to replace the field with the AccomplID of the 'school accompanist'. To do this, we execute the following query: SELECT AccomplID FROM ExamAccomps WHERE School = True. We have already carried out validation at this point to ensure no more than one field exists where 'School' is true. If the result returns an AccomplID, this AccomplID will replace the word "school" in the array. If the result returns nothing, an error message will be displayed to say that the school accompanist has not been selected. If the field is anything other than 'none' or 'school', we need to check if it is an accompanist's email address. As an accompanist's email address is stored in Supervisors, we need to join the tables together. We cannot just search through Supervisors, because the email given could be a supervisor who is not an accompanist in this exam cycle. And in this situation, an error must be returned. The following query is executed: SELECT AccomplID FROM ExamAccomps JOIN Supervisors ON ExamAccomps.AccomplID = Supervisors.SupervisorID WHERE Email = \$email. \$email is the inputted data. This query joins the 2 tables together whenever the AccomplID and SupervisorID are equal. This means that the email is compared to every accompanist's email in the current cycle, **no one else**. If an AccomplID is returned, this is stored in the array, replacing their email address. If not, then the email is either not a registered accompanist, or there is a spelling mistake in the input. This error is displayed on screen.

If all validation checks have been passed, every row in the array should now contain: StudentID, Instrument, Grade, SupervisorID, AccomplID. This is now inserted into Examinees (ExamineeID = StudentID). If errors are found at any point in this process, the specific error message is displayed, along with the data that caused the error. This helps Ms Pearcey to locate and correct the mistake easily. All of this validation is necessary, as any mistakes in the foreign key prevent queries from being executed later in the program.

### SCHEDULE

Field name	Data type	Validation	Attributes	Notes
ScheduleID	INT	None	PRIMARY KEY, Auto-increment, Not null	

ExamineeID	INT	None	Not null	FK for Examinees
StartTime	TIME	None	Not null	

Figure 2.4.1L: SCHEDULE table and validation

This table contains the created schedule. The creation of this table is a major point in the program, and is outputted to all users in some form. This table has an auto-incrementing primary key called ScheduleID. This ID allows each record to be uniquely identified. I could have made ExamineeID a primary key, but future iterations of this program may involve editing the examinee information **after** publishing. In this case, the primary key would have to be altered, which poses significant challenges. When the schedule has been created, the 'edit' function recalculates the StartTime for each record as records are moved. The order of records in this table does not change, but the order of exams does. In order to display the table in chronological order, the command ORDER BY StartTime can be used. The StartTime and ExamineeID are obviously needed in this table, to ensure who's exam it is, and what time the exam is scheduled for. The StartTime will be stored in MySQL's data type "TIME". The format for this is hh:mm:ss.

All of these fields are necessary, but any other fields would prevent this table from being in 3NF. The ExamineeID is a foreign key, creating a one-to-one relationship between Schedule and Examinees. Therefore, each record in Schedule can be linked to every other table through the foreign keys in Examinees. For example, I can obtain the Accompanist's FirstName and LastName for the first record in Schedule like so:

```
SELECT FirstName, LastName
FROM Supervisors
JOIN ExamAccomps ON Supervisors.SupervisorID =
ExamAccomps.AccompID
JOIN Examinees ON ExamAccomps.AccompID = Examinees.AccompID
JOIN Schedule ON Examinees.ExamineeID = Schedule.ExamineeID
WHERE ScheduleID = 1
```

For the first entry into Schedule, this query will return the examinee's accompanist's first and last name. This is just an example, but through the foreign key of Schedule.ExamineeID, the schedule can be linked to everything in the database! Therefore, any other fields would be transitive dependencies. For example, the instrument is dependent on the examinee, which is not a primary key. Therefore, Instrument cannot be a field in 'Schedule'. It is worth noting that Examinees.AccompID is the only field that can be null in the database. Therefore, I may have to use LEFT JOIN, RIGHT JOIN, and FULL JOIN when dealing with this field to get the desired result. In this table there are no repeated attributes, no transitive dependencies, and no many-to-many relationships. This table, just like every other table in this database, is in third normal form. By constantly improving upon my design to get to this point, I have simplified my workload later. Normalised databases are easier to maintain, easier to query, more flexible and less redundant. This efficient database design also minimises the data that Ms Pearcey will have to enter, simplifying the scheduling process and making it easier to use.

## 2.4.2 Key variables and data structures

As section 2.4.1 shows, most of my data is stored within the database. This database is clearly the main data structure in my program. Therefore, the majority of my variables and arrays will be used to store data in the database, or to hold outputs of queries to the database. An example of this is the array created from the CSV of examinee information, which is used as a middleman between the CSV file and the Examinees table. Nevertheless, there are some key variables, arrays and objects that are crucial to the solution.

Name	Data type	Purpose
examineeIDs	Array of integers	Stores the ExamineeID's of all unscheduled examinees during the scheduling process. If the array is empty, the schedule is complete.
time	Object of PHP's DateTime class	Stores the 'current' time as exams are scheduled, used as input into Schedule.StartTime, and to ensure the schedule follows TCL rules. It is an object of the DateTime class so that it is easy to increment the time and store the StartTime.
sessionTime	Integer variable	Stores the number of minutes in the current session (cannot exceed 120). Used to ensure the schedule follows TCL rules.
dayExamTime	Integer variable	Stores the total number of minutes of exams in the whole day (cannot exceed 390). Used to ensure the schedule follows TCL rules.
interval	Integer constant	The maximum time difference (in minutes) from the standard break time within which a break can be scheduled. This is around 10 minutes, but iterative testing will allow me to find an optimal value for maximum scheduling. Used to determine when to execute the 'squeeze' function.
numCompleted Breaks	Integer variable	Number of completed breaks, used to determine what time the next break is. This ensures the timetable will follow TCL rules and will work towards the ideal break times.
eod	Boolean variable	Determines if it is the end of the day. Used to make sure exam fits TCL regulations, eg. no exams are scheduled past 5pm.
accompIDs	2D array of integers	Used in the recalculateOrder function. Stores the free time of each accompanist at a certain time, to find which accompanist needs to be scheduled the most urgently.

records	2D array	Used in the ‘squeeze’ function. Stores examineeID and Duration of unscheduled examinees who have the current accompanist or no accompanist. Array is used to find the exam with the best duration that takes the time closest to the ideal break time.
conn	Object of MySQLi class	Stores the connection to the database, allowing the program to read from and write to the database.

Figure 2.4.2a: Table of key data structures and variables

None of this data needs validation. All of this data is either initialised by the computer program, or taken from the database. At this point in the program, the database has already been validated very carefully, which has been explained in section 2.4.1. For example, examineeIDs will take data from the ExamineeID field in the Examinees table. As this is already validated to be an integer, it does not need to be validated again. Other data, such as the sessionTime variable, is created by the program and changed by the program. Therefore, none of the data listed in the table above requires validation (as a result of meticulous validation when the database is filled).

## 2.5 Algorithms

### 2.5.1 Rethinking the project's scale

Throughout this project, I have tried to be ambitious with the range of features and functionality in this software. I originally believed that the currently designed solution was achievable within the time and resources available. However, many modules in this solution have started to become increasingly complex and difficult to implement over time. As I tried to design the algorithms for each section of my structure diagram, I started to understand that parts of my solution required extensive algorithms in order to implement them effectively. If I attempt to continue designing and developing my full solution within this timeframe, the overall project quality will be hindered, and essential components of the solution may be compromised. The challenges I faced when developing these detailed algorithms have made me reconsider the scale of this project.

It is important that this project does not continue to expand in its complexity, therefore certain parts of the project must be removed. I decided to talk to Ms Pearcey in person, discussing which aspects of the project are most important to her. She is my client, and therefore I will try to meet her requirements for this project. By directly discussing with her I can get a clear understanding of which aspects of the solution are essential to her, and which are not. One mistake I want to avoid is trying to take on too much work, and eventually neglecting the most important elements of the project.

My discussion with Ms Pearcey was in person, and we discussed at length about which features are most important, and which can be removed. The key idea she stressed is that the main schedule function must work. This means that it takes a valid (full) database, and produces a valid schedule that meets the requirements. Eventually, we created a list of features in order of priority, which is shown below.

*List of features by priority, created by Ms Pearcey and I:*

- Develop schedule function (takes a validated database to produce schedule). This includes:
  - Function to ensure schedule meets TCL regulations
  - Grouping exams appropriately
  - Meeting accompanist availability
  - Scheduling breaks at correct times (according to Ms Pearcey's specifications)
  - Recognising invalid inputs (eg. insufficient availability / too many examinees)
- Produce PDF of schedule
- Develop accounts:
  - Login and registration for all users
  - Organiser inputs initial information (start date)
  - Organiser can enter examinee information as CSV
  - Organiser can input accompanist information
- Advance login functionality:
  - Users shown specific and relevant information of timetable
- **Removed:**
  - Deleting schedules at 10pm on the day of the exam

- *Develop 'edit' function to allow edits to the schedule before publishing*
- *Accompanists enter their own availability*
- *Organisers can see status of accompanists while waiting for availability*
- *All email functionality*
- *Re-edit and Re-publish feature*
- *Ability for organiser to view/create/delete accounts*
- *Organiser can create accounts in bulk through CSV files*

This list shows that some parts of the project have been removed completely, and will not be designed anymore or developed. Below, I will explain and justify each aspect of this list to provide some insight into the priority of each element. I have also linked each feature to the structure diagram from section 2.2.1 to clarify how the structure of the solution has changed.

**The schedule function - structure diagram 2.1:** Ms Pearcey emphasised that this function is a key requirement. This is the most important part of the project, as it carries out the key process of exam scheduling. Ms Pearcey made it clear that I should focus on this first, and prioritise the success of this part of the solution over everything else. The project, after all, has always aimed to reduce the workload for Ms Pearcey, and this function's success will make this possible. This will be one single function, called 'schedule', but due to the level of complexity it will have several other functions to improve modularity and structure of the code. This feature will include some more complex validation, such as checking if a schedule is breaking regulations, or if accompanists have entered insufficient availability.

**Produce PDF - structure diagram 3.3:** An important part of the solution is to produce a PDF of the final schedule. This feature can prevent the tedious task of Ms Pearcey manually creating a PDF file so she can stick it on the music notice board. As the recent examinee survey ([1.3.2](#)) shows, 95% of examinees are familiar with the process of checking their time slot on this notice board. By automatically creating this PDF file, the new system can integrate easily into the school without confusing anyone.

**Login and registration functionality - structure diagram 1.2:** This feature involves developing a system that registers users into the database, and takes each user to their dashboard upon login. Again, lots of validation is required. It enables the examinee and accompanist information to be linked to key data such as the person's name. This feature is very important, as it populates the table Students and Supervisors, and allows each user to see their dashboard. However, it is not as essential to the scheduling process as the others. The system can still schedule exams without this data, but it will yield better results if this data is available.

**Organiser inputs initial information - structure diagram 1.1.3:** This feature, again, provides an interface for Ms Pearcey to enter important data into the database. This data will be validated to ensure it is sensible, then used later in the scheduling process. This feature is essential, as it allows key data to be inputted.

**Enter examinee information CSV - structure diagram 1.3:** While the schedule function will read from a database, the CSV file of examinee information will provide essential data to populate this database. This CSV file is easy for Ms Pearcey to create, so my system should be able to validate the file and enter it into the database.

**Organiser enters accompanist information - structure diagram 1.4.3:** This feature is also very important, as it provides a simple interface for the organiser to input essential data into the database. Without it, the database would have to be populated manually. In this feature, the organiser will select an accompanist and enter the data for them. Ms Pearcey finds that it is more important for her to enter this data, which is why this feature has a higher priority than accompanist's entering their own data. This way, Ms Pearcey said that she has control over what goes into the database, which reduces the chance of mistakes.

**Users view relevant information - structure diagram 3.1 (and 1.1.1):** This feature allows all users to see the time slot for their exam on the website. It abstracts irrelevant information and allows them to view the information from anywhere as soon as it is published. Ms Pearcey said that this feature would be ideal, as it will benefit all users in the system, providing a central location for everyone to view the schedule as soon as it has been published. This feature also includes Ms Pearcey's ability to view the entire schedule once the process is complete.

### *Removed features*

**Deleting schedules at 10pm on the exam day - structure diagram 1.1.2:** This feature involves deleting the schedule and its data at the end of the exam cycle, which maintains the database and allows more schedules to be made. This is clearly important, as otherwise only one schedule could ever be made by the system before having to manually fix the database. However, I have run into several problems with how I can schedule a script to run at a certain time and day. This has caused me to rethink how the project will maintain the database properly. Instead, I will design a button that allows Ms Pearcey to start a new schedule. This button will delete the relevant tables to maintain the database. ***Starting a new schedule will delete the existing schedule and its data,*** and this button will be available for Ms Pearcey to click at any time. This feature will allow multiple schedules to be made, maintaining the database to achieve exactly the same goals as originally planned.

**The edit function - structure diagram 2.2:** All the included features either make the schedule, or enable the schedule to be made by managing data. Achieving those features will ensure that Ms Pearcey can enter data through a simple interface and receive a PDF of the created schedule. This feature allows Ms Pearcey more freedom and flexibility, making it important but not essential to the key aspect of this project. We agreed that, while this is a useful feature, it would become quite complex within the project's time constraints. This could prevent me from developing the more fundamental parts of the solution, and therefore this feature has been removed.

**Accompanists enter their own availability - structure diagram 1.4.1/4:** This feature allows the database to be populated by the accompanists. However, Ms Pearcey is already comfortable with handling availability herself, and will likely have to pester these accompanists to submit their availability on their own. This also could introduce some errors into the database, forcing Ms Pearcey to handle it. Ms Pearcey's workload is not likely to be affected by this feature, so its development is not important to her. Therefore, in the interest of focussing on features that are important to Ms Pearcey, this feature has been removed.

**Organiser can see accompanist status - structure diagram 1.4.1.1:** As the feature above has been removed, this feature also must be removed. The revised process for entering accompanist availability is: a list of registered supervisors will be displayed. By clicking on their name, Ms Pearcey can enter accompanist availability for that particular user. Submitting this data will redirect her back to the first page, where she can enter more data or continue to the next step.

**All email functionality - structure diagram 1.2.1.2 , 1.4.2 , 3.2 , 3.4:** The decision to remove this feature was quite difficult. As I started to design the full algorithm to email the timetable to everyone, I discovered several major issues. This involved the difficulties of using the PHPMailer library to schedule emails, as the one possible solution could only work if the website was open at the correct time, and would require a much more complex database structure to store the scheduled emails. Other solutions require using paid services such as AWS, and I do not currently have the resources to implement this solution. Therefore, email scheduling was removed.

However, there was a much larger concern regarding the security of my system. In order to send emails, they have to be sent from an email account. As Ms Pearcey is the administrator and organiser, I planned to use her account to send all emails, which would allow her to respond to any email replies. However, I later learnt that, in order to send these emails, Ms Pearcey would need to input her password into the system. Although I could build a system that stores her plain text password as a variable, and uses it to send emails, this can raise serious security risks. My system has very little security measures in place to make it less susceptible to malicious attacks. Therefore, if someone gains unauthorised access to the server, Ms Pearcey's password could be compromised. This could lead to severe consequences and further sensitive personal information could be accessed using this password. It is not safe for anyone to have access to email passwords, including me, and can raise serious concerns related to the Data Protection Act of 2018.

Therefore, I tried to find another solution. Instead, I could use OAuth for authentication. This involves the website displaying 'sign in with google', using APIs to connect to Google's secure servers. This method would be secure and PHPMailer supports OAuth, but would be quite complex for me to understand and implement effectively. By implementing this method into my code, I would be forced to neglect other, more important parts of my project (such as the schedule function). Therefore, due to security issues and concerns about complexity, all email-related functionality has been removed. I informed Ms Pearcey about some of these issues, and she agreed that the emails are absolutely not necessary, as they do not contribute to the scheduling of exams. I hope to incorporate email functionality into this system in the future, but it is not within the scope of this project.

**Re-publish feature - structure diagram 2.3:** As other aspects of my solution have become increasingly complex, I have decided to remove this feature. When I initially spoke with Ms Pearcey about cutting the project down, this was one of the first things she suggested. This feature is quite complex, and forces other essential functions such as the publish function to support this level of complexity. If this feature was included, there is no guarantee that it will ever be used. It would help the organiser if problems occur, but does not assist with the scheduling process any more than the 'edit' function. This feature would make several other

features more complex in order for the whole schedule to be re-published. Due to the scale of the project as well as the time and resources available, I have decided not to proceed with designing and developing this feature.

**CSV account creation and ability to manage accounts - structure diagram 1.2.1.3:**

These features have been removed in order to simplify the accounts functionality, as it was becoming increasingly difficult to manage. After considering the motivation of this project, and which features create the most impact on the success of the solution, I concluded that this feature was the least important. During my discussion with Ms Pearcey, I found that she felt the same way, and was not bothered about this feature. Although it could make the process slightly more efficient, its development has proven to become considerably more complicated than originally planned. I now understand that this feature is not within the scope of the project, as my primary focus is to ensure the scheduling functionality produces efficient and valid schedules.

The features that have been removed will no longer be implemented in the remaining part of the project. Therefore, this document will not mention these features from now on. Although the features have been removed, I hope to implement them in the future. Therefore, I will continue to collect the same data, using the same database structure. During development, I use this list of priority to ensure that Ms Pearcey's requirements can be achieved to the best of my abilities.

To clarify the structure of my new solution, I have changed my structure diagram to indicate removed features in red. This aims to emphasise which features have been removed, and which are remaining, so the structure of my solution remains clear. This updated structure diagram can be seen below, with red boxes indicating that the feature has been removed. Box 1.1.2 is highlighted in green as it has been changed in this section.

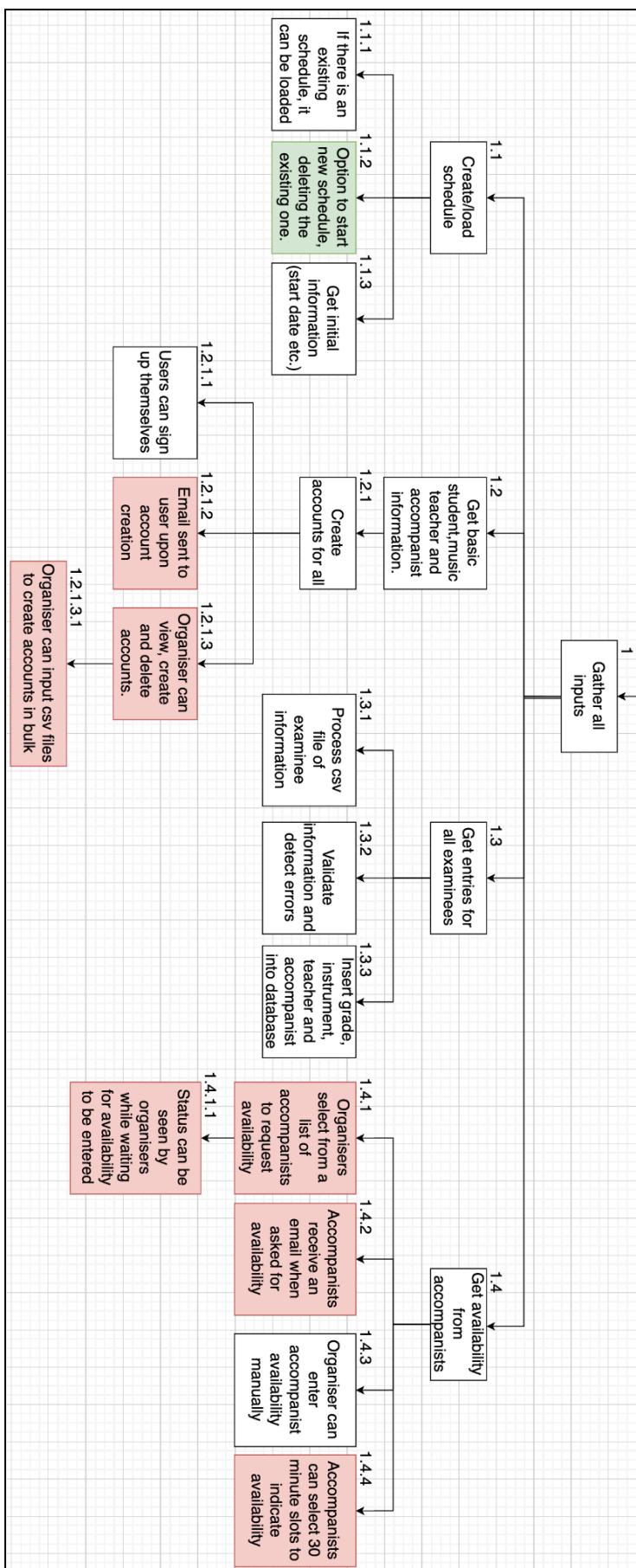


Figure 2.5.1a: Updated structure diagram (1) - Gather All Inputs

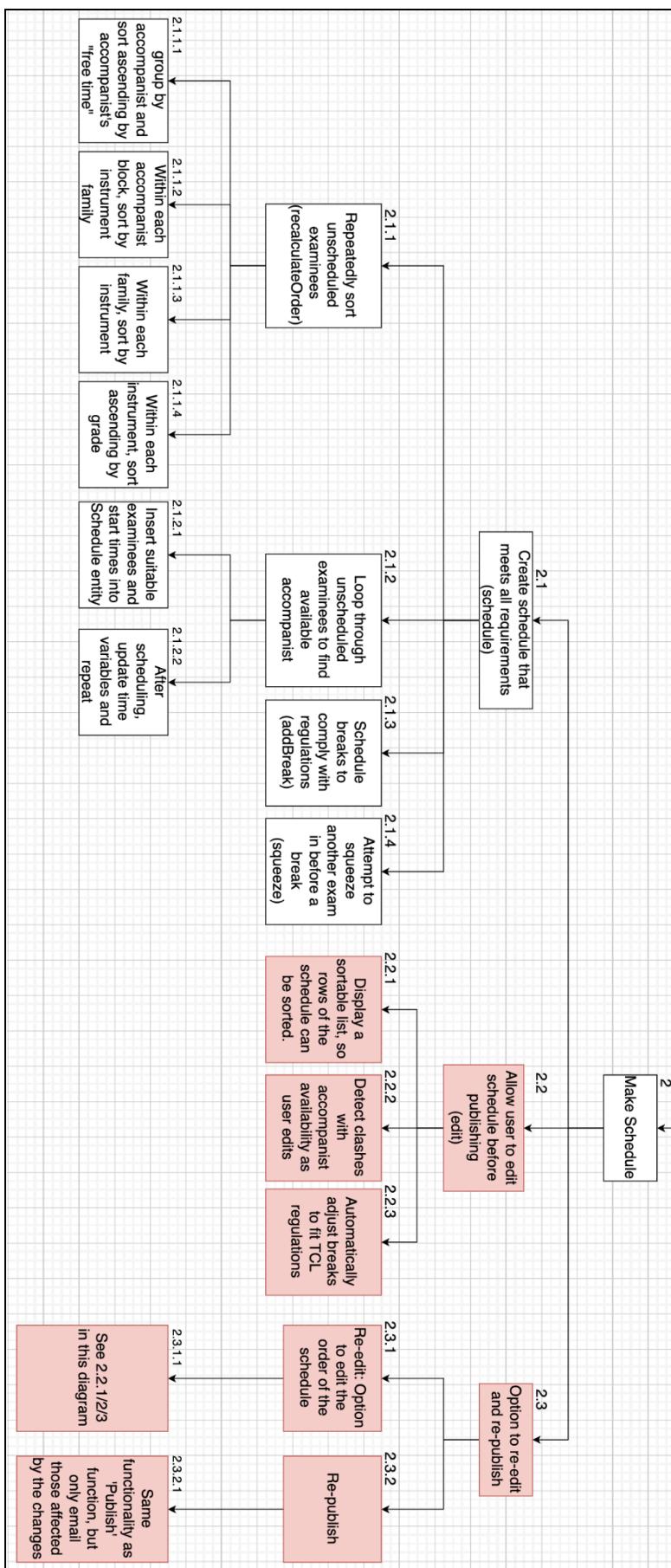


Figure 2.5.1b: Updated structure diagram (2) - Make Schedule

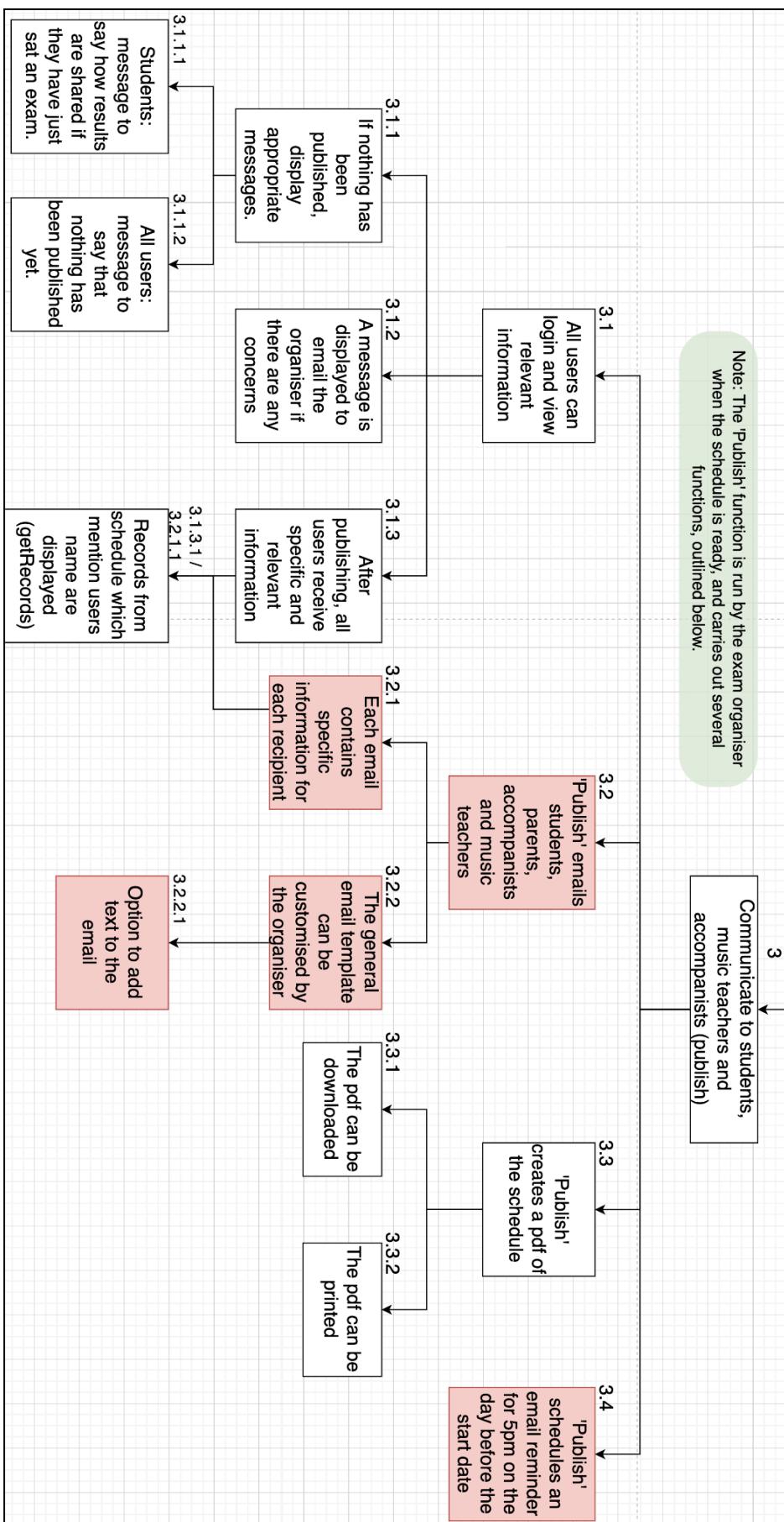


Figure 2.5.1c: Updated structure diagram (3) - Publish Schedule

## 2.5.2 The schedule function

The most important part of this project is the schedule function, which is why it has been designed first. It must take validated data from the database to produce a schedule that meets all regulations and schedules exams efficiently, meeting success criteria S1-8. The function is very complex, and therefore has many other functions associated with it to allow the feature to be designed fully. The flowchart for the ‘schedule’ function is below.

# SCHEDULE()

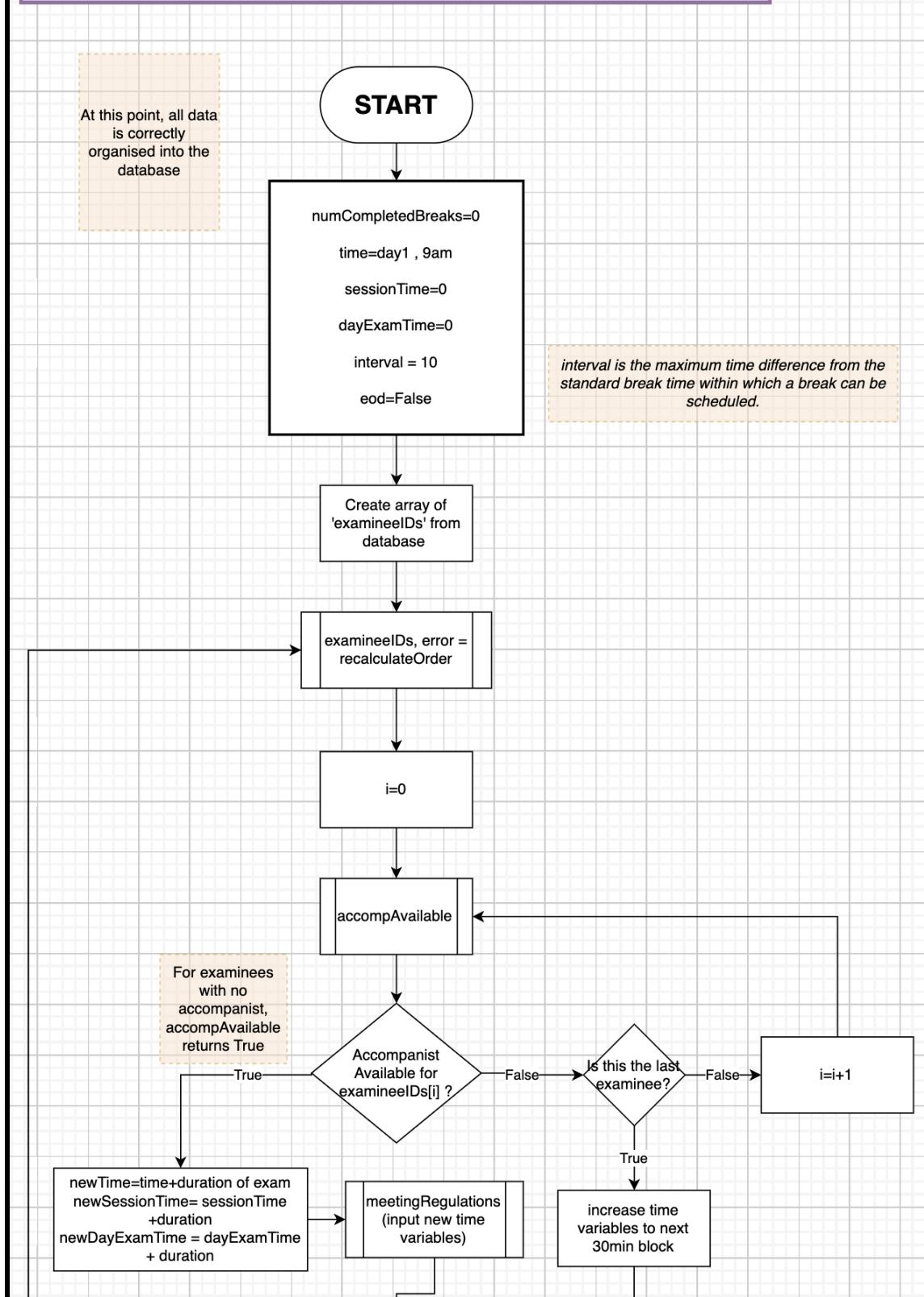


Figure 2.5.2a: schedule function - 1

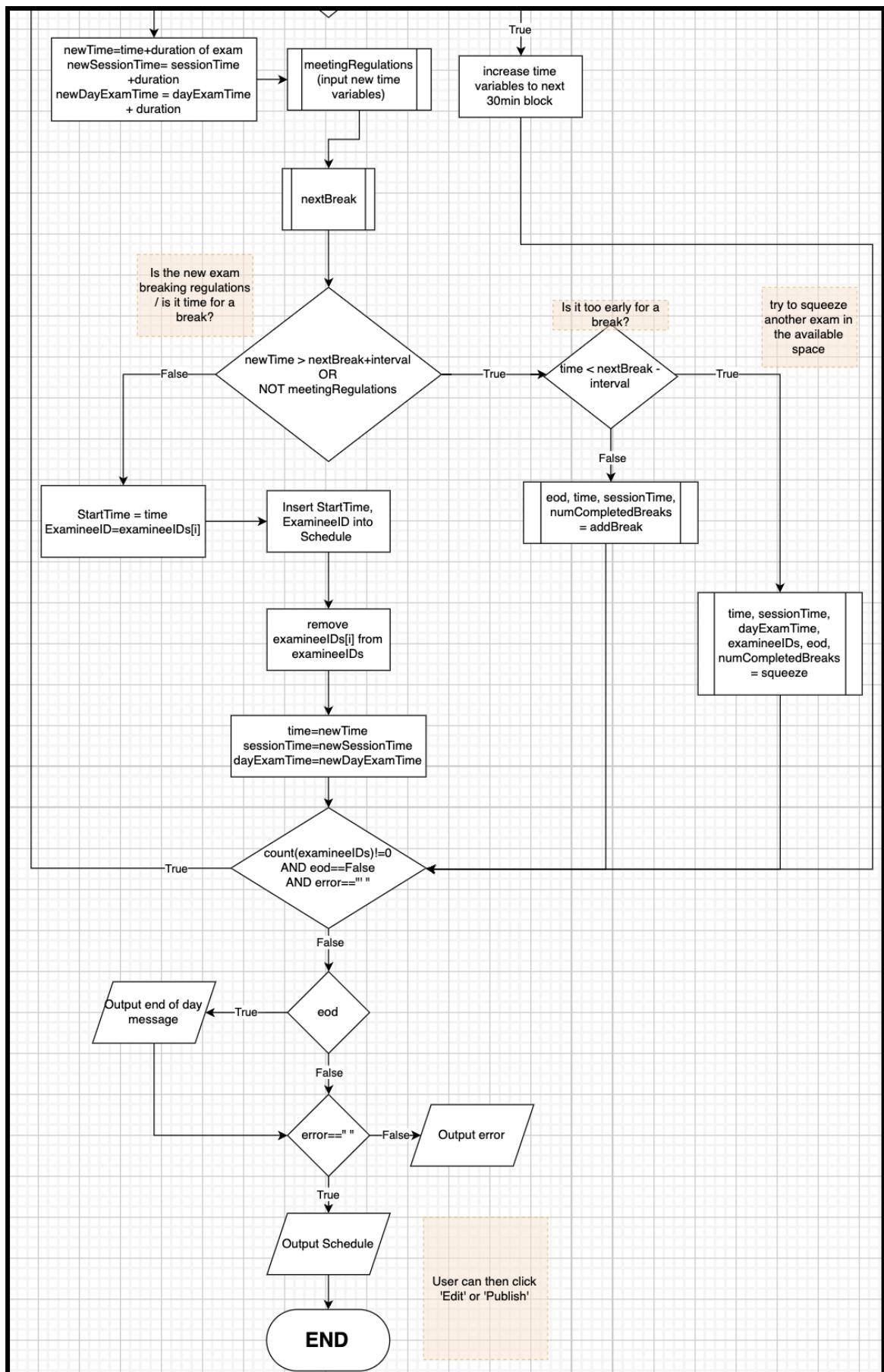


Figure 2.5.2b: schedule function - 2

### *schedule()*

After initialising variables, the algorithm will retrieve data from the Examinees table to make the array examineeIDs. This array contains the IDs of all examinees that have not yet been scheduled (which is initially everyone). Then, the function recalculateOrder will return the array of examineeIDs in order of priority. Therefore, the first element in the list should be scheduled with the most urgency. The function schedules exams and breaks, incrementing the ‘time’ object each time something is scheduled. This object stores cumulative elapsed time in order to track the progression of the schedule as it is developed. This, as well as other variables, allow the algorithm to decide if a break should be scheduled or not, as well as other key decisions. recalculateOrder runs each time the ‘time’ object changes, to ensure that the most important exams are always scheduled.

The array examineeIDs will have the most urgent exams first, so by looping through this list from the start until an available accompanist is found, we can ensure that the best choice of exam is made. If no accompanists are available (and all exams require an accompanist), the time is incremented to the next 30 minute block and the loop is repeated. I could have incremented the time by 1 minute to be more sensitive, or calculated when the next accompanist is available by querying data from the database. However, if no accompanist is available, and availability is recorded in 30 minute blocks, then no accompanists will be available in the current block. Therefore, the most efficient method is to simply move on to the start of the next block.

Once an exam with an available accompanist is found, the algorithm must check if this new exam is breaking regulations in any way. The key realisation I had is to understand that all problems with TCL regulations can be solved by scheduling a break. For example, if the new session time is exceeding 2 hours, forget this new exam and schedule a break. This brings the session time down to 0, solving the issue. Therefore, the algorithm will calculate the new time variables, adding on the duration of the new exam. Then, if these new time variables are breaking regulations, or if they are too far past the ideal break time, a break is scheduled. The function meetingRegulations returns true/false if the new times are meeting/breaking regulations, and nextBreak simply returns the ideal time for the next break, as per Ms Pearcey’s request. The function nextBreak is relatively simple, and could be implemented here. However, I chose to create a separate function because it is very possible that Ms Pearcey may decide to change the ideal break times, so that they could align more closely with school break times. In this situation, only the nextBreak function has to be edited. As the function is used many times across different algorithms, it ensures consistency throughout the system, if code is not repeated it cannot be inconsistent.

A break should be scheduled if the new time is “too far past the ideal break time”. This statement is the same as ‘newTime > nextBreak + interval’. The interval is a boundary that determines how strictly we want to follow these break times. If the interval is low, the algorithm will be more determined to schedule breaks at the correct time. If it is high, it will be more lenient, and give more importance to grouping exams appropriately. Originally, I designed this algorithm so that breaks were scheduled immediately when concerns were raised. However, if a new time is breaking regulations, it does not ensure that the old time is within the interval for a break. Remember that the algorithm should try its hardest to schedule breaks within the interval of the ideal time. Therefore, I included an additional

check. If ‘time < nextBreak - interval’, the algorithm finds that it is too early for a break, so it should try to squeeze in one more exam before scheduling a break. This is the purpose of the squeeze function. This function will squeeze in another exam, but if it cannot a break will be scheduled. This ensures that the algorithm tries to maximise the number of exams scheduled, and minimises the amount of time wasted, helping the success criteria S2 and S3 to be achieved. If the new time is raising concerns, **and** the old time is within the allowed interval for a break, a break is scheduled. This happens by calling the subroutine addBreak. (Note that ‘squeeze’ may also call addBreak if it decides to schedule a break). addBreak will also check if it is the end of the day, and if so then eod=True. The new time, sessionTime, and numCompletedBreaks are also returned.

If, however, the new time does not raise any concerns, the exam must be scheduled. This means that the ExamineeID and StartTime are inserted into the Schedule entity. Then, the examinee is removed from examineeIDs (this array only contains **unscheduled** examinees), and the time variables are updated accordingly. If the array examineeIDs is empty, there are no more examinees to schedule, so the loop is ended. However, there are other scenarios where the loop should end, even if examineeIDs is not empty.

It is important to always consider how invalid data will be handled. When designing the flowchart, I initially came up with my solution and then searched for how to catch any errors and recognise invalid data. In my algorithm many loops are utilised, which can lead to infinite loops if the condition is never met. To search for an available accompanist, a count-controlled loop is used to iterate through the array ‘ExamineeIDs’. Therefore, this loop must always end. If an available accompanist is found, an examinee is scheduled (and removed from ExamineeIDs) or a break is scheduled. If an available accompanist is not found, the time is increased to the next 30 minute slot, and the loop restarts. This larger loop is condition-controlled, and originally would only end if ExamineeIDs is empty. As I have shown, there are only 2 scenarios where an examinee is not scheduled:

1. A break is scheduled
2. No accompanists are available at that time

If one of these scenarios is to occur repeatedly, examinees will never be scheduled. Therefore, the function must detect the issue and return an error message to prevent an infinite loop.

For a break to be scheduled, the next exam’s duration must be in the interval of possible break times. This is a problem if a break is to be scheduled repeatedly. Therefore, for this to occur repeatedly, the duration of an exam would have to exceed the duration of a single session (120 minutes). In simpler terms, a 121 minute exam would cause this algorithm to search through the entire day to look for a time slot, and the invalidity of this entry would not be detected. However, the table ‘Timings’ is static data, and we can clearly see from figure 1.1.2b that the longest exam is 32 minutes. Therefore, scenario 1 can be ignored.

In scenario 2, no accompanists are available. This can happen for several reasons. Note that the accompanist for every examinee has already been validated to exist in ExamAccomps. In any situation, if freeTime is **ever** negative for any accompanist, then an accompanist’s exams will take longer than the time they are available for, so scheduling becomes impossible. In this case, some examinee(s) will never be scheduled, so

ExamineeIDs will never be empty. Therefore, I decided that in the function 'recalculateOrder', selection must be used to identify if freeTime is negative. If this ever happens, then the program must stop and return an error message, detailing which accompanist has caused this issue. This selection will detect problems, no matter what may be causing accompanists to be unavailable.

After Ms Pearcey and I agreed to remove the functionality of scheduling across multiple days, I adjusted the flowchart so that the addBreak function can detect if the end of the day has been reached. As a result, the condition-controlled loop must end if eod=True, or if error!=" ", or if ExamineeIDs is empty. This ensures the loop will never continue infinitely, and instead will return the appropriate messages. At the end of the function, the schedule will be displayed.

Scheduling exams has been created as a single function to make the system more usable, as this one function will be run by a highlighted button on screen. This simplifies the interface, abstracting all unnecessary information and preventing excessive inputs, which can frustrate the user.

Several subroutines have been called by schedule(), all of these have been designed using flowcharts below.

### *recalculateOrder(examineeIDs, time)*

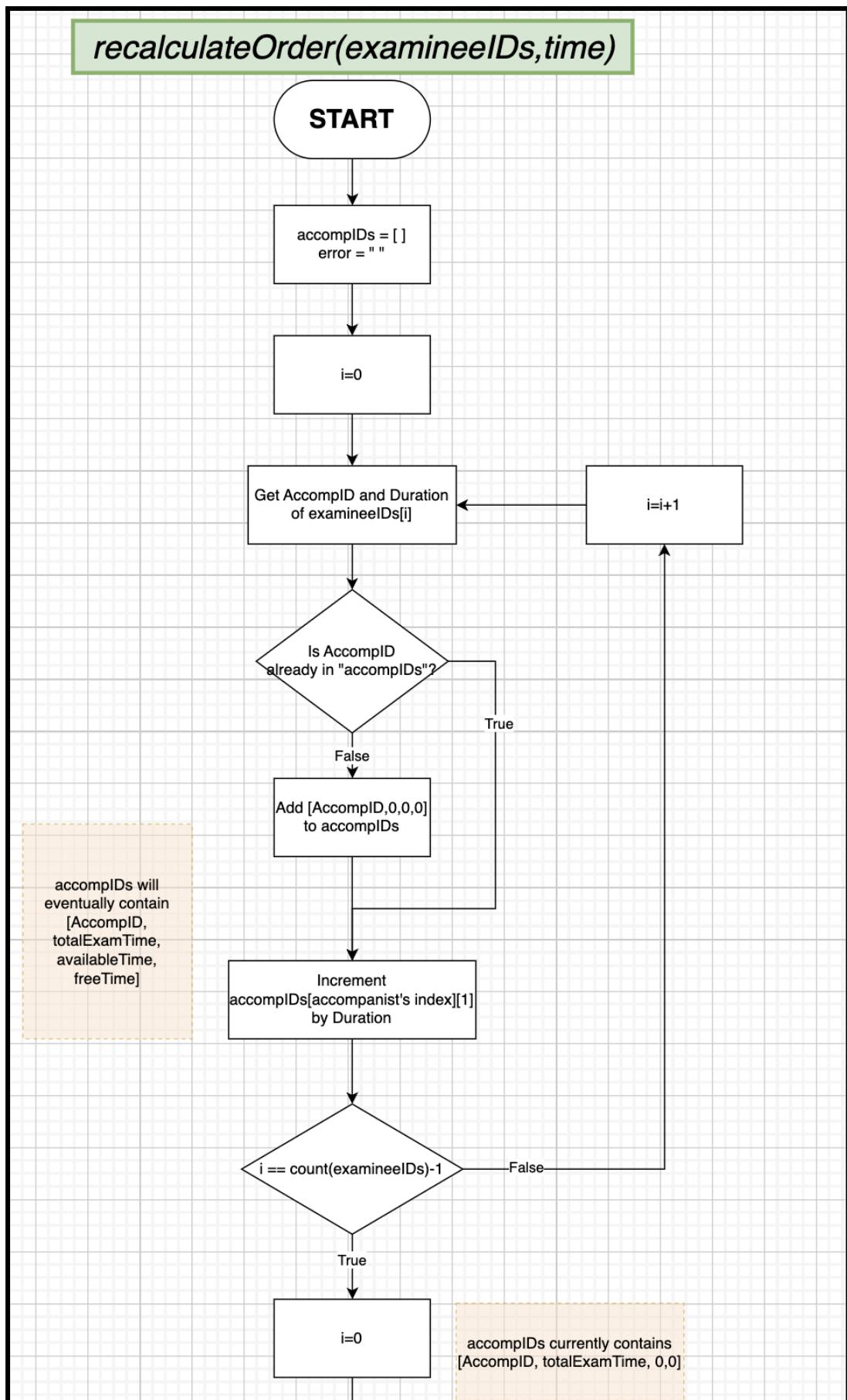


Figure 2.5.2c: *recalculateOrder* - 1

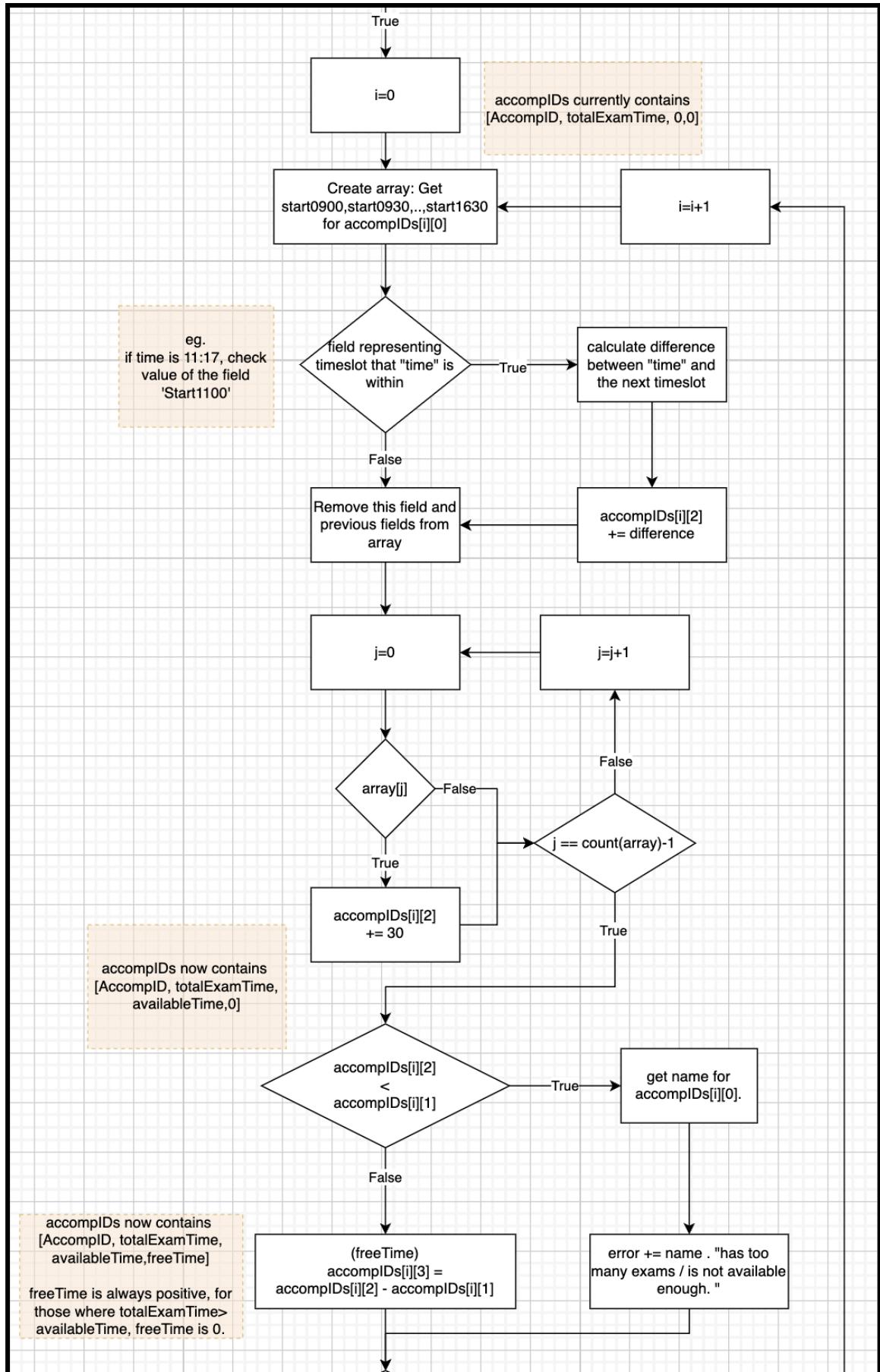


Figure 2.5.2d: recalculateOrder - 2

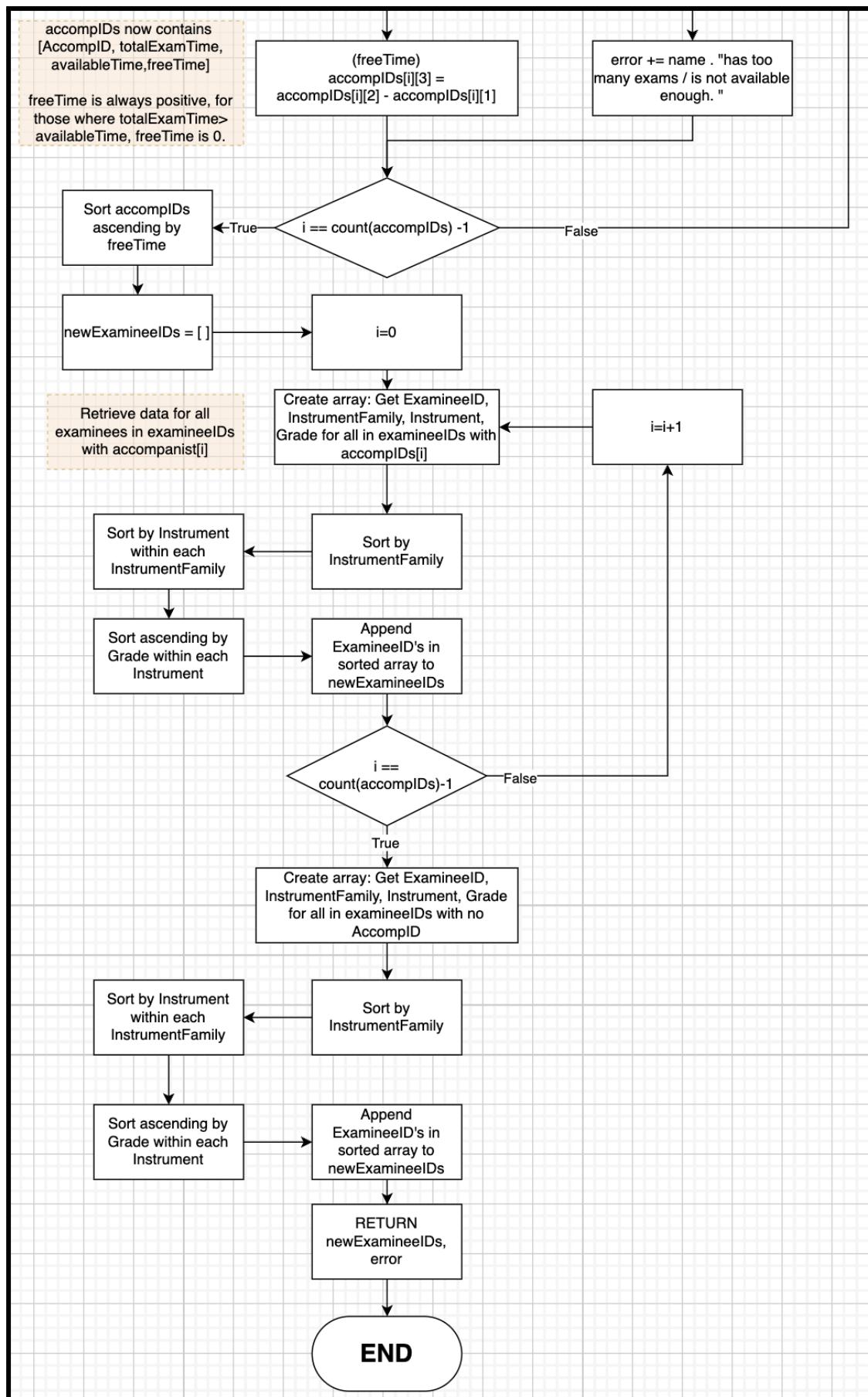


Figure 2.5.2e: recalculateOrder - 3

### *recalculateOrder(examineeIDs,time)*

This function was the most difficult to design, and is the key to grouping and prioritising exams effectively. I researched different pre-existing scheduling algorithms, but due to the specific requirements of the project I had to develop my own algorithm. As exams should be sorted and grouped in a very specific way, I soon realised that the best method to do this involved creating an array of examineeIDs. Simply put, this function will sort this array so that the exams with the highest priority are first. It will also sort exams into groups to ensure the TCL requirements are met, and will return an error if there is a problem.

The difficulty in scheduling these exams stems from the fact that they must work with every accompanist's availability. Therefore, this availability should be used to determine which exams need to be scheduled first. If we have a list of examinees in order of priority, we can find the first examinee with an available accompanist and schedule them (if they meet the TCL regulations). This is the purpose of *schedule()*. My thought process was then figuring out how to create this list.

The list should primarily depend on the accompanist's urgency. But how can I quantify this 'urgency'? Originally, I thought each examinee should be sorted based on their accompanist's total number of exams. But each exam has a different duration, so it should be sorted by total duration of exams. This method seemed like it could work, but I later realised that this did not consider availability, and only tried to schedule accompanists who were used more. This method was not sensitive to availability. This algorithm would fail if an accompanist with very few exams and very limited availability, and another accompanist with many exams was always available. I need this list to prioritise accompanists who have limited availability. This pushed me towards sorting examinees by their accompanist's total minutes available. However, this soon became clear to me as a poor choice, as an accompanist with limited availability would be too heavily prioritised. This method was too sensitive to availability.

The best solution I could find was a result of assessing these 2 attempts, and trying to find a middle ground between the two. This solution has now been designed, and I believe that it correctly decides which accompanists are more urgent than others. In this method, examinees are sorted by their accompanist's free time. An accompanist's free time is calculated by subtracting the duration of all their exams from the time they have available, or:  $\text{freeTime} = \text{availableTime} - \text{totalExamTime}$ . If an accompanist has lots of exams, but is available a lot, they will have a large freeTime. If an accompanist has few exams, and is available the same amount as the first accompanist, they will have a larger freeTime. If an accompanist has many exams, and is not very available, they will have a low freeTime. By sorting examinees ascending based on their accompanist's freeTime, they can be sorted correctly by urgency! Those with no free time will be prioritised over others with more free time, so that accompanists can be scheduled in such a way that everyone's time is used as productively as possible.

This list of urgency is already being grouped into accompanist blocks, which is a separate requirement by Ms Pearcey. As one accompanist has one freeTime, all examinees with the same accompanist are grouped in the list. This means that each accompanist's exams will be in groups, so that their time is not wasted. Within each accompanist block, TCL

regulations specify exams to be grouped by instrument family, then instrument, then sorted ascending by grade. The function will then carry out these processes to make sure that the list meets regulations as well. It is worth noting that these groupings are seen as ‘ideal’, and are not as strict as other rules. Therefore, accompanist availability will inevitably cause exams to be scheduled out of these groups, but by creating an ideal list, we can keep encouraging the groups to be formed.

As examinees are scheduled, and time is incremented, accompanist freeTime changes. Therefore, to ensure the most urgent exams are always prioritised, this function is called at the start of each loop. This means that the list is re-ordered whenever something changes. Additionally, if an accompanist’s freeTime is ever negative, then this accompanist has to play in more exams than they are available for. This is a serious issue, and the function recalculateOrder will return an error to inform Ms Pearcey which accompanist(s) is/are causing the issue.

In order to design the algorithm, the first step is to calculate freeTime for each accompanist. The 2D array ‘accompIDs’ will eventually contain [AccompID, totalExamTime, availableTime, freeTime]. Remember that this function must only take into consideration the examinees in examineeIDs, not all examinees. Therefore, the algorithm loops through examineeIDs, adding the duration of the exam to the relevant accompanist. This calculates totalExamTime for each accompanist.

Then, availableTime must be calculated. This is the total time an accompanist is available for **after** the current time. Eg. if an accompanist is available from 9am-10am and then from 2pm-3pm, their availableTime at 1pm is 60 minutes. At 2:15pm, their available time is 45 minutes. A loop will run through each accompanist in accompIDs. Within the loop, the accompanist’s availability information is taken from ExamAccomps in the database. The field that represents the time slot that ‘time’ is within is checked first. If true, the difference between the current time and the start of the next time slot is added to availableTime. If not, nothing is added. Then, the subsequent fields that relate to future availability are checked (any fields before ‘time’ are ignored, as they are now in the past). If a field is true, 30 minutes is added to availableTime. This is repeated until all the fields have been checked.

freeTime = availableTime - totalExamTime , or  
accompIDs[i][3]=accompIDs[i][2]-accompIDs[i][1]. This is calculated for every accompanist, unless availableTime is less than totalExamTime. If an accompanist is available for less time than they need to play for, they either:

- Have too many exams
- Are not available enough

A variable ‘error’ will store the names of all accompanists with availableTime < totalExamTime, as well as a message to clearly express the issue to Ms Pearcey. This error is returned to schedule(), and will cause all processes to be stopped so the message can be displayed on screen. For now, any accompanists with negative freeTime have freeTime = 0.

Once the freeTime has been calculated for all accompanists, accompIDs is sorted ascending by freeTime. For each accompanist (starting from the one with least freeTime), all of **their**

examinees that are also in examineeIDs are put into a new array. This array is sorted by instrument family, then by instrument within each family, and sorted ascending by grade within each instrument. This leaves a sorted array of examineeIDs with the highest-priority accompanist. This array is added to 'newExamineeIDs', and the process is repeated for every accompanist in accomIDs.

Many examinees will have no accompanist. Throughout this function, examinees with no accompanist have been ignored. This is because these examinees can be scheduled anywhere, they do not have an accompanist so they do not need to be scheduled to fit anyone else's availability. Therefore, they will always have the lowest priority, and should be last in newExamineeIDs. All examinees in examineeIDs with no accompanist are added to a new array, which is then sorted by family, instrument and grade. This array is added to the end of newExamineeIDs, which is now finally returned to the calling function, schedule(). 'error' is also returned.

Through repeatedly adjusting and improving this algorithm, I have reached a design that creates the most efficient order of examinees, so that the accompanist's time is used as productively as possible.

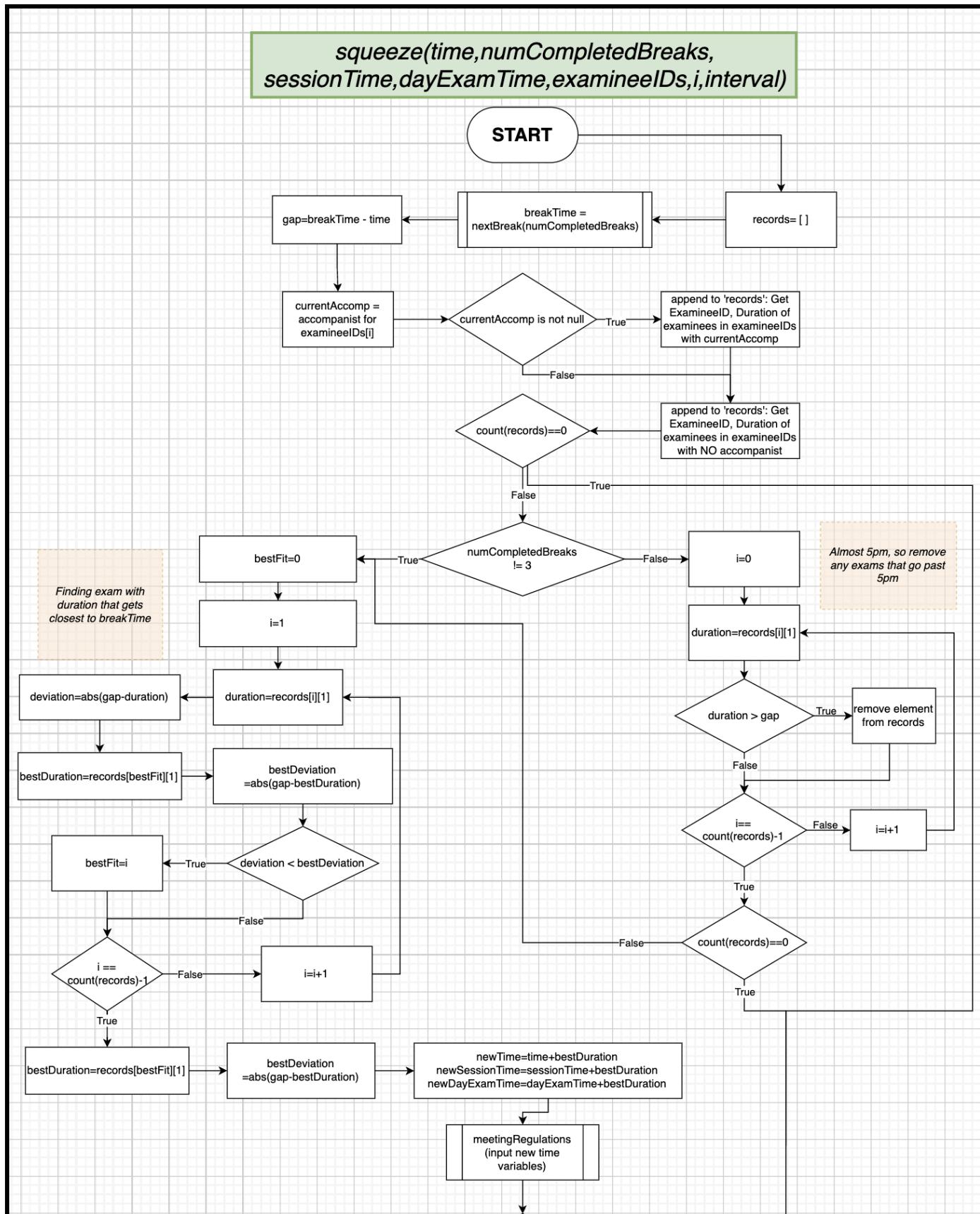


Figure 2.5.2f: squeeze - 1

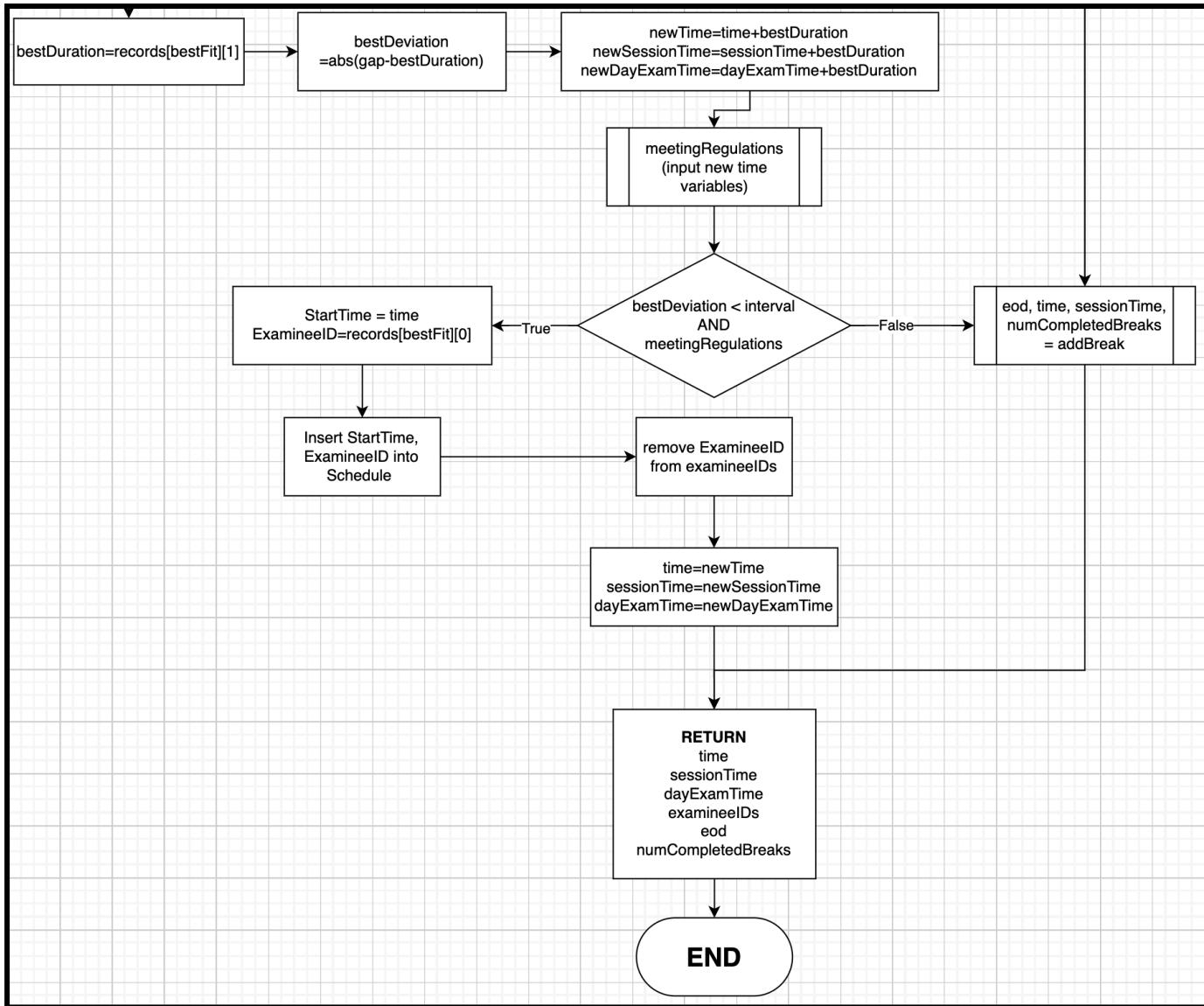


Figure 2.5.2g: squeeze - 2

`squeeze(time, numCompletedBreaks, sessionTime, dayExamTime, examineeIDs, i, interval)`

The `squeeze` function tries to squeeze another exam in before a break is scheduled. If an exam is about to be scheduled, `schedule()` will check if it is too late for an exam (and time for a break). If it is too late, it will then check if it is too early for a break. If true, `squeeze` is called. This tries to schedule one more exam that goes as close to the ideal time as possible, without severely disrupting the order of exams.

Out of all unscheduled examinees, the duration of each exam can be compared to see what takes it closest to the ideal break time. Once that exam is found, it can be scheduled. This was my initial design, however I later realised that this could force an accompanist to play just one exam at a completely different time to all of their other exams. This would force them to sit and wait for a long time, wasting their time. Ms Pearcey told me at the start of this project that she groups each accompanist's exams in order to reduce the time wasted, so that they do not have to sit around between exams. This design was clearly an issue, so I

changed it. Now, only examinees with the current accompanist, or no accompanist are considered. This means that, no matter who is scheduled, there are 3 situations:

- The accompanist is already playing, this is just another exam in their group
- The accompanist is about to start playing, this is the first exam in their group
- The exam does not require an accompanist

None of these situations inconvenience the accompanist any more than usual, therefore this method has been designed in the flowchart above. Firstly, the function calculates ‘gap’ which is the time difference between the current value of ‘time’ and the ideal break time. This ‘gap’ is the ideal duration of an exam. An exam with this duration would allow a break to be scheduled exactly at the preferred time. The squeeze function has many parameters, one of which is ‘i’. This variable refers to the index of examineeIDs which was about to be scheduled. examineeIDs[i] was going to be scheduled, which means that the ‘current accompanist’ is the accompanist for this examinee. This allows us to find the current accompanist. After they have been found, all examinees in examineeIDs with this accompanist are added to an array. Additionally, all examinees in examineeIDs with no accompanist are added to an array. This array is called records, and is a 2D array containing [examineeID,duration].

Then, there is a clear split in the flowchart. If 3 breaks have already been scheduled, we are in the final session of the day. Also, if squeeze has been called, the current value of ‘time’ is about to break TCL regulations and/or it is near 5pm. In this case, the algorithm is adjusted. This is because the time cannot exceed 5pm, this is a very strict regulation. Therefore any exams with a duration > gap are removed from consideration, as those exams would take the time past 5pm. As long as ‘records’ is not empty, the flowchart joins back with the main part of the algorithm, described below.

For every exam in records, the deviation of its duration from the ideal time (gap) is calculated. The exam with the lowest deviation, or ‘best’ duration is found, and bestFit stores this exam’s index in ‘records’. New time variables are calculated based on this duration to check if the new exam is meeting regulations (meetingRegulations). The function then runs a very similar selection to the one in schedule(). Notice that the key difference is the check if bestDeviation < interval. This checks if the deviation from the breaktime is within the interval. The interval is the maximum allowed deviation. This comparison is the **same** as checking if the time is between nextBreak ± interval in schedule(). Although they look different, squeeze checks the same criteria as schedule() to ensure consistency, and if there are no issues, an exam is scheduled in the same way as it is in schedule(). If the exam’s end time is deviating too far from the ideal break time, or regulations have not been met in any way, addBreak is called. This function schedules a break, or decides if it is the end of the day. Therefore, the squeeze function will try to search for an exam to fit in the space available, but if nothing suitable is found, a break is scheduled. It has many inputs and many outputs, as it needs to pass on the result of addBreak, as well as the updated time variables, as this function could result in a break **or** an exam being scheduled. They are all necessary to ensure the scheduling process runs smoothly.

## *accompAvailable(ExamineeID, time)*

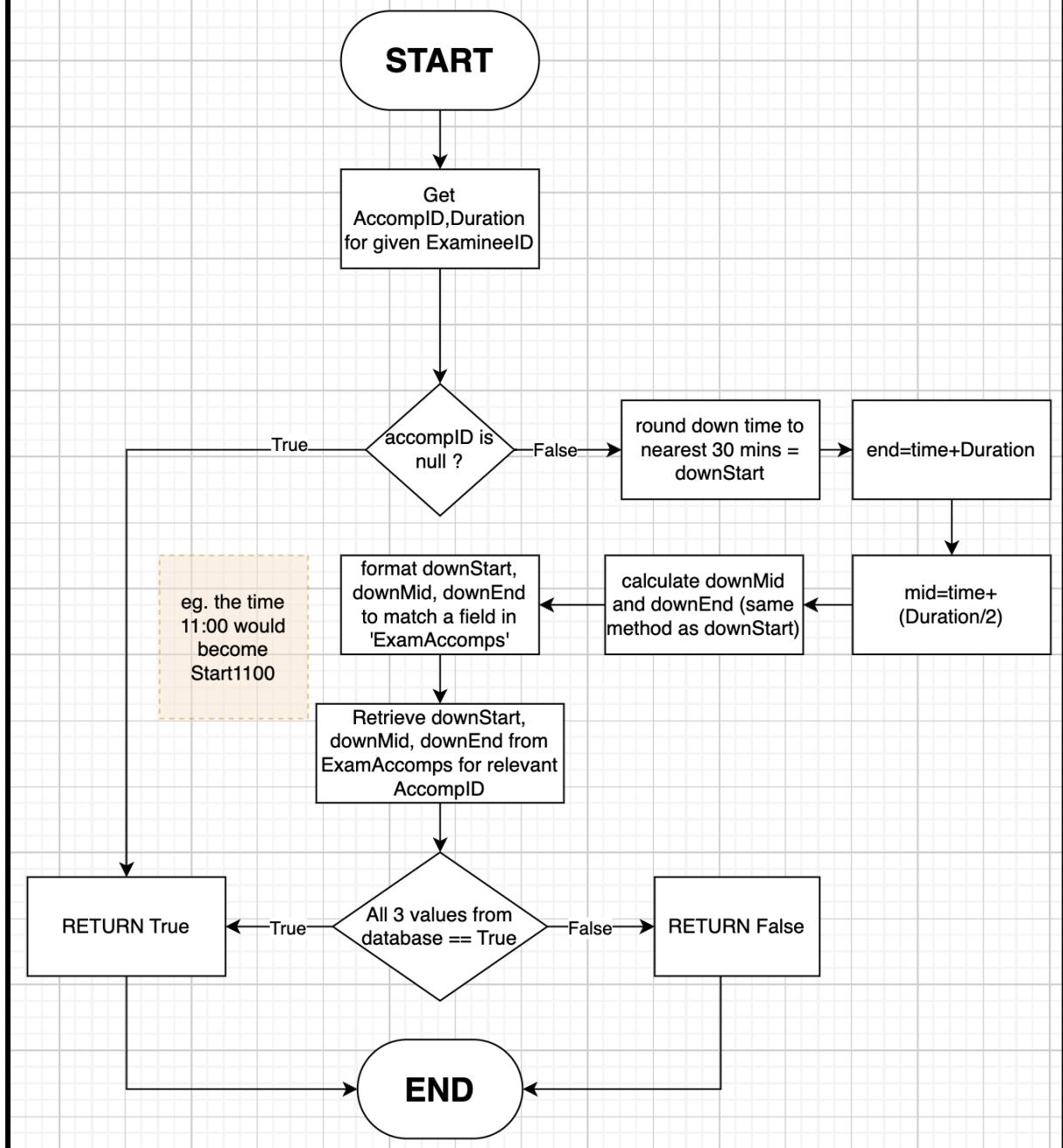


Figure 2.5.2h: accompAvailable function

## *accompAvailable(ExamineeID, time)*

This function is used to check if an examinee's accompanist is available during a certain time. Although there are more rigorous ways to check availability, I have used a relatively simple method here. This is because an exam can have a maximum duration of 32 minutes, and accompanist availability is recorded in blocks of 30 minutes. Therefore, at the very most, an accompanist would need to be available for 3 slots to be available for an exam. Although extremely unlikely, it is possible that a 32 minute exam is scheduled for 10:59am, for example. This exam ends at 11:31am, and the accompanist would have to be available

during the entire time. This corresponds to checking if Start1030, Start1100, and Start1130 are all true in the ExamAccomps table. In order to check this, you could take the start time, midpoint, and end time of this exam. Then, check the corresponding fields in the database for each. This situation is extremely unlikely, but every other situation involves shorter exam durations than this. Therefore, every single situation can be solved with this method. Although it is slightly unconventional, it confirms that the accompanist is available during the entirety of the exam.

An SQL query finds the AccomplID and Duration for this examinee. First, if the examinee does not have an accompanist, the function returns True. This allows schedule() to start scheduling this examinee. If they have an accompanist, it calculates the middle and end times of the exam based on the time and the duration, (start time = time). Then the start, middle and end times are all rounded **down** to the nearest 30. This is because accompanist availability is recorded as Start0900, Start0930, Start1000, etc. If Start0930 is true, an accompanist is available from 09:30 to 10:00. Working backwards, if you have a specific time, rounding it down to the nearest 30 minutes will help you to find the corresponding field in ExamAccomps. Once the times have been rounded down, they are formatted to match a field in ExamAccomps (so 11:00 would become Start1100). An SQL query is used to select the result of these fields for the accompanist. If all fields are true, the function returns True. If not, it returns False. It is very likely these three ‘fields’ may all be one field in the database. For example, a 15 minute exam starting at 09:12 will only require Start0900 to be checked. In this case, my function will try to retrieve Start0900 three times from the database. This is slightly redundant, but still arrives at a correct solution. Therefore, my algorithm must have support for the probable scenario where these 3 fields are repeated.

## *nextBreak(numCompletedBreaks)*

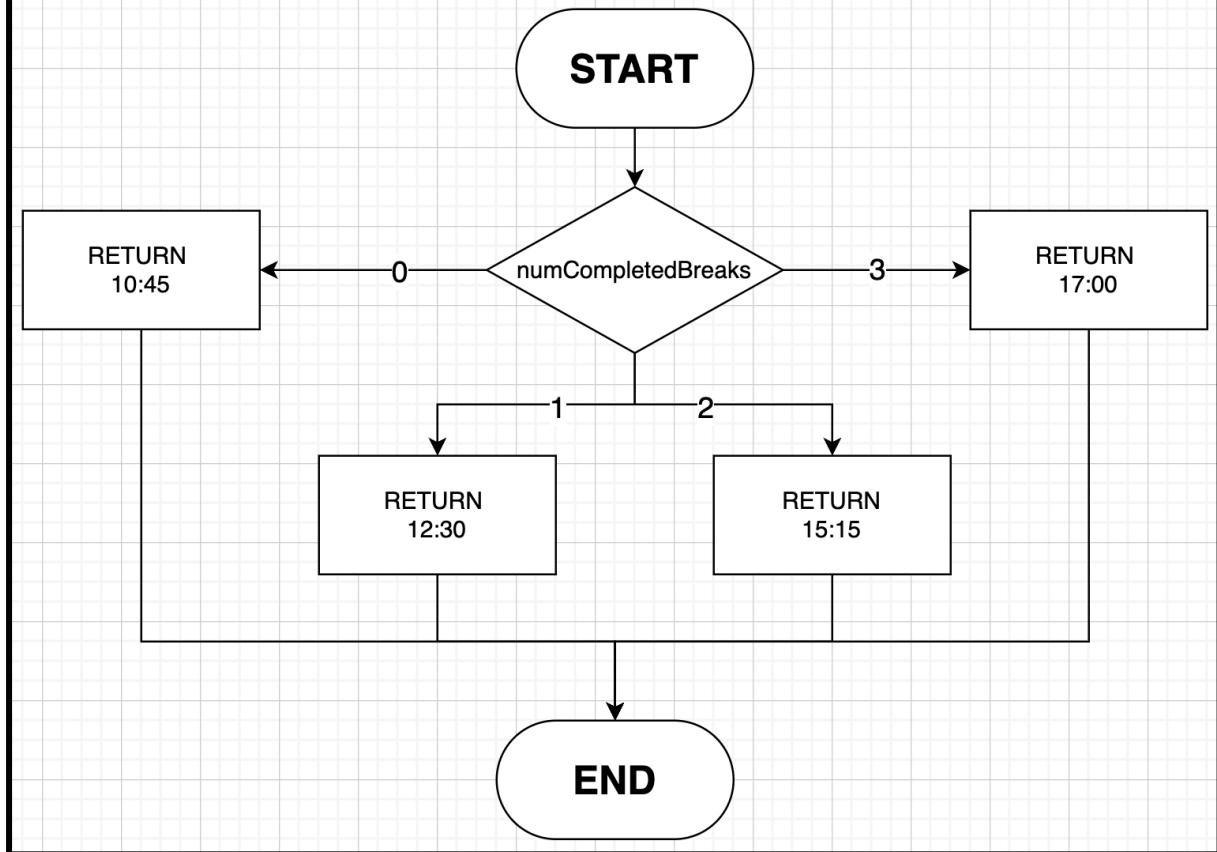


Figure 2.5.2i: *nextBreak* function

## *nextBreak(numCompletedBreaks)*

This function is very simple, but is used repeatedly throughout the entire scheduling process. Therefore, to prevent repeated code and possible inconsistency, I have created a separate function for it. Based on the number of breaks that have been completed, the function returns the ideal time for the next break to occur. These times have been specified by Ms Pearcey, after I contacted her to discuss further about the project. It is highly possible that these times may change in the future, for a number of reasons. Due to the modular nature of this scheduling process, I would only have to change the time here, and the entire scheduling process would be updated. This simplifies any future maintenance of the project. It will return the time using PHP's `DateTime` class, which is the same class used by the `'time'` object. Often, the returned break time is compared with `'time'`, so using the same class makes it easier to make these comparisons. Although 5pm is not a break, this time is returned to reduce complexity in the `schedule` function **without** breaking any regulations or diminishing the quality of the algorithm.

## *meetingRegulations(time,dayExamTime, sessionTime,numCompletedBreaks)*

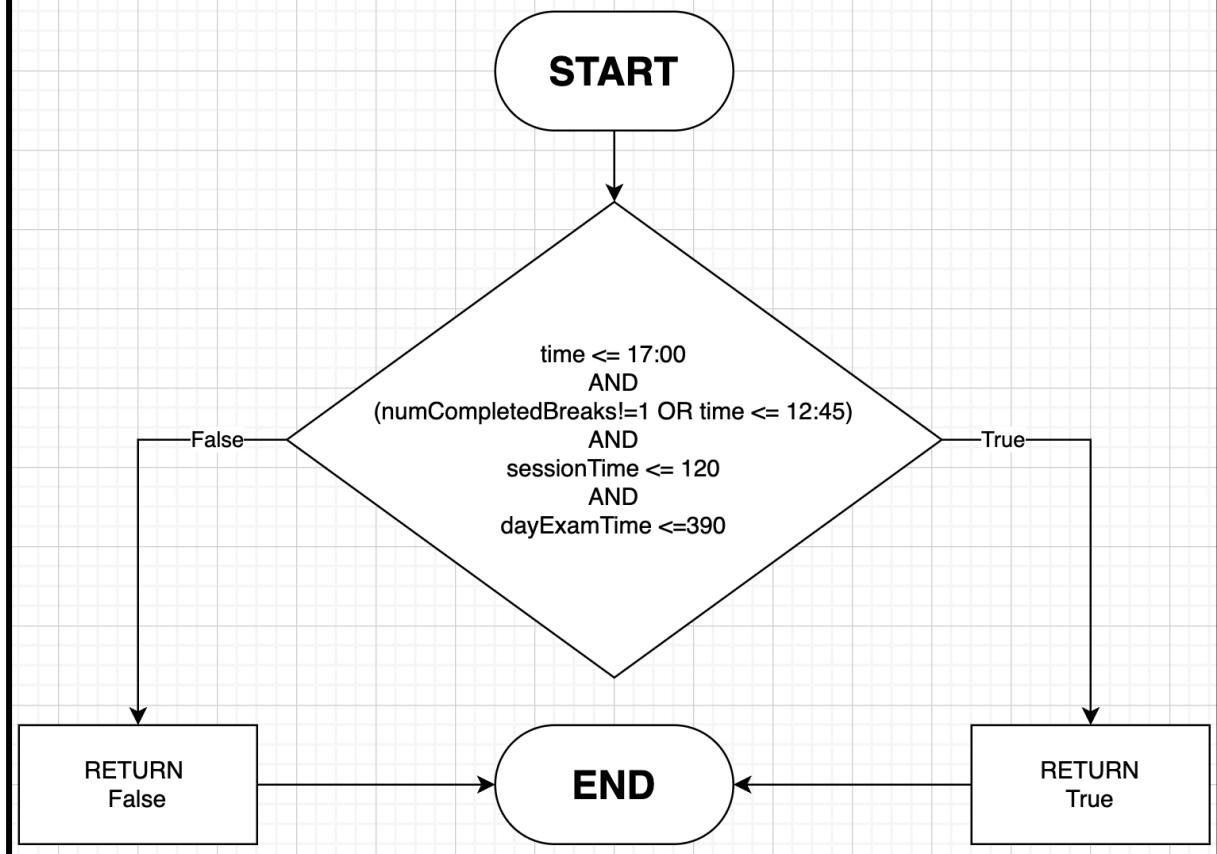


Figure 2.5.2j: meetingRegulations function

## *meetingRegulations(time, dayExamTime, sessionTime,numCompletedBreaks)*

This function is used to determine if the current state of the schedule is meeting the TCL regulations. These regulations are strict, and unlike other rules there is no leniency which allows them to be defined in the structure as shown above. The rules, which have been specified fully in section [1.1.3](#), take a number of parameters. These are all variables used to keep track of different values, their purpose has been clarified in [2.4.2](#). The first condition is clear, the time must not be past 5pm. Because exams in total cannot last more than 8 hours per day, and they start at 9pm, this is fixed at 5pm. The second condition is more complex and relates to the rule: A one hour lunch break must be timetabled after no more than 3.5 hours (not including breaks). If numCompletedBreaks is 1, a lunch break has not been scheduled yet. If it is 0, the time is too early, and if it is 2 or 3 a lunch break has already been scheduled. Therefore, if numCompletedBreaks is 1, we need to check if it has been more than 3.5 hours (not including breaks). As the time starts at 9am, 12:45 is the latest a lunch break can be. This means that, if numCompletedBreaks is 1 **and** it is past 12:45, the regulations are not being followed. The converse of this condition is

`numCompletedBreaks!=1` or `time <= 12:45`. If this statement is true, regulations are being met so far. The converse has been used in the flowchart to make sure that true conditions relate to the result being true, to ensure consistency and simplicity in the function. The next 2 conditions are clear: each session must be a maximum of 2 hours (120 mins) and the maximum exam time is 6.5 hours (390 mins) per day (not including breaks). If `sessionTime <= 120` and `dayExamTime <=390`, and the previous 2 conditions are true, the current schedule is meeting regulations, so `True` is returned. If any of the conditions are false, `False` is returned as the regulations are being broken. All data in the databases is already validated before the schedule function is run, but the purpose of this function is to validate the current status of the schedule against TCL rules. This function details all validation carried out as the schedule function runs.

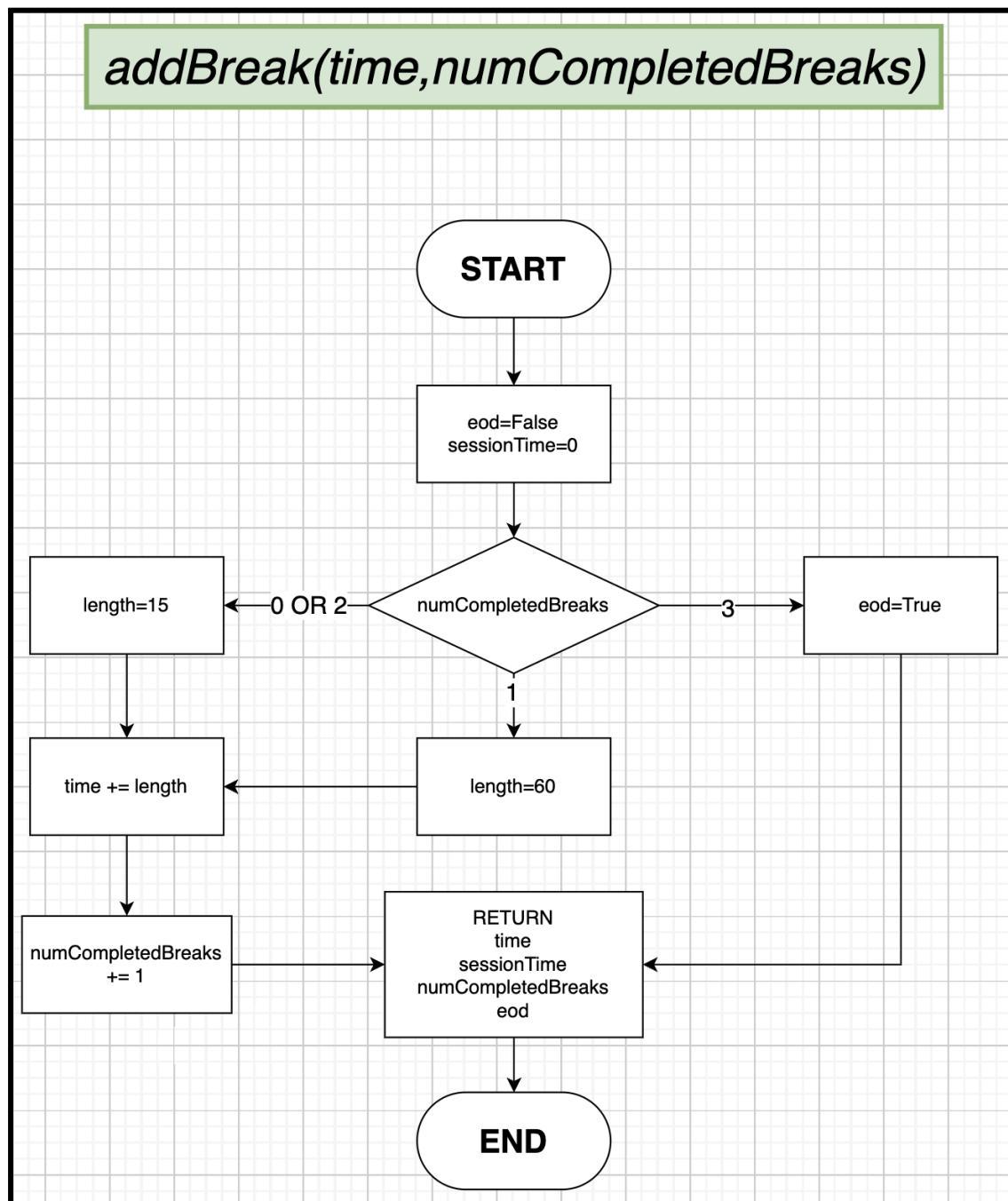


Figure 2.5.2k: addBreak function

*addBreak(time,numCompletedBreaks)*

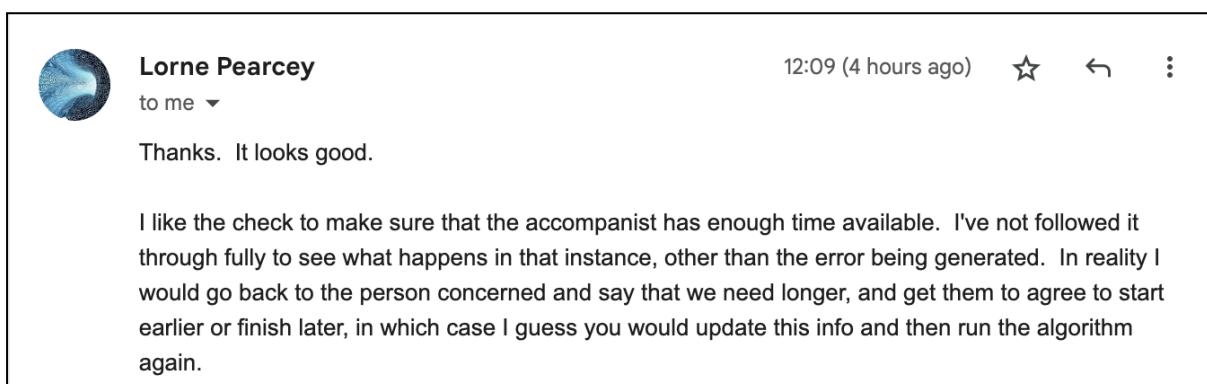
This function is called by schedule() and squeeze() to increment the time appropriately, or to signify the end of the day. It does not need to check if it is time for a break, this is done by the calling function. If it is being run, it must be time for a break. If numCompletedBreaks is 1, it adds 60 minutes to the current time as it schedules a lunch break. If numCompletedBreaks is 0 or 2, it adds 15 minutes to schedule a normal break. Note that the Schedule entity is not affected, but it will be made clear as the next exam to be scheduled will be later than the previous exam's start time + duration. numCompletedBreaks is incremented to show that the break has been scheduled.

If it is being run, it is also possible that it is the end of the day. Note that nextBreak() can return 5pm, and meetingRegulations returns false if the time is past 5pm. These functions form the are involved in the selection that decides whether or not to call addBreak, and have been designed carefully so that if it is time for a break, or time to end the day, addBreak will be called. Therefore, if numCompletedBreaks is 3, all breaks have been scheduled, so it must be the end of the day. Therefore, the boolean variable eod is set to True.

No matter what happens, if the function is called, the current session has ended. Therefore, sessionTime = 0 is returned. As sessionTime is always returned as 0, it does not need to be passed as a parameter. Also, time, eod, and numCompletedBreaks are returned to update the schedule of the changes made. This function updates the schedule's current state by adding a break or signalling the end of the day. The calling function can then act accordingly, for example it will stop the loop if eod=True, and output the schedule along with a message to say that the end of the day has been reached.

### **Feedback on scheduling algorithm**

This is the most important part of the solution, as Ms Pearcey and I agreed in section 2.5.1. Its success is crucial to the success of this project, as it details the exact method by which exams are scheduled. To confirm that this design met all of Ms Pearcey's requirements, and so that we were on the same page, I sent the above flowcharts to Ms Pearcey. She has been scheduling music exams for over 20 years and is very knowledgeable about how exams should be scheduled, and is also my client for this project. Therefore, her feedback and opinion is crucial to me as I aim to meet her needs. She replied with the following email:



Lorne Pearcey  
to me ▾

12:09 (4 hours ago)

Thanks. It looks good.

I like the check to make sure that the accompanist has enough time available. I've not followed it through fully to see what happens in that instance, other than the error being generated. In reality I would go back to the person concerned and say that we need longer, and get them to agree to start earlier or finish later, in which case I guess you would update this info and then run the algorithm again.

### Figure 2.5.2L: Ms Pearcey feedback on scheduling algorithm

As you can see, she was happy with the functionality provided by the scheduling algorithm. She especially liked my design of `recalculateOrder()`, this function checks that all accompanists have enough time available, and uses this principle to prioritise some exams over others. If an accompanist does not have enough available time, an error message is displayed on screen. Ms Pearcey then went on to say how she would use this error message to come to a solution. This is a great sign, showing she is already visualising how this system will be implemented, and is becoming familiar with its features. Not only do the designed algorithms meet her requirements, but they are also capable of integrating easily with Ms Pearcey's methods. As we were both pleased with the algorithm's design, I did not make any further changes to this function.

This section has fully designed section 2.1 in the structure diagram.

### 2.5.3 Login system

An important part of the solution is the login system. Registering a user gives the database essential information for the schedule to be created. Logging in gives Ms Pearcey exclusive access to the scheduling functionality. It also allows specific and relevant parts of the schedule to be displayed, abstracting any data that is not important to the user. The interface for logging in and registering will be quite simple to prevent confusion, and the login page has been designed in [2.3.1](#). The login page will be called 'login.php'. When registering a user, students should be entered into the 'Students' table, and supervisors should be entered into the 'Supervisors' table. These 2 tables are quite similar, with the only difference being that 'Students' has one extra field: the parent's email address. This difference means that I cannot simply have one page for registering users, there must be one for students and one for supervisors.

Therefore, clicking 'register' on the navigation bar will take the user to 'register.php', a very simple page displaying 2 links. This page will give users the option to register a student, or to register a supervisor. This will take the user to 'register-student.php' or 'register-supervisor.php'. These 2 pages, as well as login.php, will all require validation to ensure that all data entered is sensible, and to prevent future errors. As the 2 registration pages are very similar, I have developed pseudocode for registering a student only. The pseudocode for registering supervisors is the exact same, but all validation of parent emails is removed, and the data is inserted into Supervisors rather than Students.

```

## REGISTRATION | register-student.php (/ register-supervisor.php)
if submit button is clicked
    firstName = get first name and sanitise input
    lastName = get last name and sanitise input
    email = get email and sanitise input
    parentEmail = get parent email and sanitise input
    password = get password and sanitise input
    confirmPassword = get 2nd password and sanitise input

    if any field = ""
        print("You must enter data for all fields")
    else
        if length(firstName) <= 50 and length(firstName) >= 2 and firstName has no special characters and no numbers
            if length(lastName) <= 50 and length(lastName) >= 2 and lastName has no special characters and no numbers

                email=filter_var(email,FILTER_VALIDATE_EMAIL)
                parentEmail=filter_var(parentEmail,FILTER_VALIDATE_EMAIL)
                if email or parentEmail is false
                    print('Invalid email/parent email')
                else
                    if email in Students table
                        print("Email already registered as a student. Please log in")
                    else
                        if email in Supervisors table
                            print("Email already registered as a music teacher / accompanist. Please log in")
                        else
                            if password != confirmPassword
                                print('Passwords do not match')
                            else
                                if length(password)<8 or password has no numbers,no uppercase, or no lowercase letters
                                    print("Password must be at least 8 characters long and include at least 1 uppercase
                                         letter, 1 lowercase letter, and 1 number.")
                                else
                                    hashedPassword=hash(password)
                                    Insert following data into Students: firstName,lastName,email,hashedPassword,parentEmail
                                    redirect to login.php
                                endif
                            endif
                        endif
                    endif
                endif
            endif
        else
            print("Last name must be 2-50 characters, with no numbers or special characters (except hyphens)")
        endif
    else
        print("First name must be 2-50 characters, with no numbers or special characters (except hyphens)")
    endif
endif
endif

```

Figure 2.5.3a: Registration pseudocode.

## Registration

This validates the entered data as specified in section [2.4](#). The inputs are collected and sanitised, which will likely use the function `mysqli_real_escape_string` to prevent SQL injection. Then, a presence check is carried out on all fields. If any fields are empty, an error message is displayed. The first and last names are validated to check that:

- Name is at most 50 characters
- Name is at least 2 characters
- Name has no numbers
- Name has no special characters (except hyphens)

If this check is failed, a relevant error message is displayed informing the specific problem to the user. This message is informative and should help them to understand how to fix this error.

Then, emails are validated against RFC 822 regulations. In this case, the pseudocode uses the PHP function `filter_var`, along with the argument '`FILTER_VALIDATE_EMAIL`'.

<code>FILTER_VALIDATE_EMAIL</code>	"validate_email"	<code>default</code>	<code>FILTER_FLAG_EMAIL_UNICODE,</code> <code>FILTER_NULL_ON_FAILURE</code>	Validates whether the value is a valid e-mail address.
				In general, this validates e-mail addresses against the addr-spec syntax in » <a href="#">RFC 822</a> , with the exceptions that comments and whitespace folding and dotless domain names are not supported.

Figure 2.5.3b: The filter\_var function with 'FILTER\_VALIDATE\_EMAIL' checks against RFC 822.

As I wanted to validate against the rigorous RFC 822 syntax, I searched if there was a pre-existing function to do so in PHP. This led to me finding this function, the screenshot above shows that RFC 822 is used in the logic of this function. If regulations are not being met, it returns false. If they are, it returns the valid email address. Therefore, the email and parentEmail are rewritten by running this function, and an error message is displayed if either are false.

A user cannot be registered twice. They cannot be registered in both tables either. Therefore, the provided email is checked to see if it is already registered in the Students or Supervisors table. This occurs regardless of whether a student or supervisor is being registered. If this check is passed, the 2 passwords are checked to verify that they are the same. This minimises the risk of spelling mistakes and verifies that the correct password will be registered.

The password must also meet regulations to make sure that it is strong, adding a level of security to the system. The following checks are carried out:

- Must be at least 8 characters
- Must have at least 1 uppercase letter
- Must have at least 1 lowercase letter
- Must have at least 1 number

If valid, the password is finally hashed using a secure hashing algorithm. Hashing is a deterministic one way process, so there is no way to obtain the password from its hash. Therefore, the hash is stored in the database. This protects the user's privacy even if someone gains unauthorised access to the database. Once the relevant data is entered into Students/Supervisors, the user is taken to login.php.

```

## LOGIN | login.php
if submit button is clicked
email = get email and sanitise input
password = get password and sanitise input

if email=="" or password==""
    print('You must enter a username and password')
else
    if filter_var(email,FILTER_VALIDATE_EMAIL) == false
        print('Invalid email')
    else
        StudentID,students_password = get ID and password from Students table for given email
        SupervisorID,supervisors_password = get ID and password from Supervisors table for given email
        hashedPassword = hash(password)
        if students_password != null
            if hashedPassword=password
                // LOGGED IN
                session variable 'user_type' = 'student'
                session variable 'user_id' = StudentID
                redirect to dashboard.php
            else
                print('Incorrect password')
            endif
        elseif supervisors_password != null
            if hashedPassword=password
                // LOGGED IN
                session variable 'user_id' = SupervisorID
                if email = 'admin@email.com'
                    session variable 'user_type' = 'admin'
                else
                    session variable 'user_type' = 'supervisor'
                endif
                redirect to dashboard.php
            else
                print('Incorrect password')
            endif
        else
            print('Email not found')
        endif
    endif
endif
endif

```

Figure 2.5.3c: Login pseudocode

## *Login*

Once users have registered, they can log in to view information about the schedule. To prevent SQL injection, all inputs are sanitised immediately. If either field is empty, the presence check will fail and an error message is returned: ‘You must enter a username and password.’ The email is also validated using the same function as before.

The correct password for this given email could exist in Students or Supervisors. Ms Pearcey’s email and password is registered in Supervisors, as she is a music teacher as well as the administrator. First, the password that is linked to the given email is retrieved from both tables. As no one can register in both tables, at least 1 of these results is empty, and the stored password will be null. If the student password is null, the supervisor password is checked. If this is also null, an error message is displayed: ‘Email not found’. The password is also hashed using the same hashing algorithm used to store the password.

If the email is found in Students, students\_password will not be null. This will contain the hash of the correct password, so it is compared to the hash of the given password. If they

are different, the password is incorrect. If they are the same, the user should be logged in. The session variable `user_type` is set to ‘student’, `user_id` is set to their StudentID and they are redirected to the dashboard, `dashboard.php`.

If found in Supervisors, there is an additional check. If the password is correct, the program checks if the email is ‘`admin@email.com`’, because this is the admin login. If so, then the admin has correctly logged in, so the `user_type` is set to ‘admin’. If not, it is set to ‘supervisor’. `user_id` is set to their SupervisorID, and they are taken to the dashboard as well.

There are 2 session variables being used here. Session variables are transferred across pages, and will be used throughout the system to ensure the relevant information is displayed to the intended users. One important benefit of the `user_type` is that it gives the admin exclusive access to scheduling features, as users with other `user_types` will be redirected to another page.

Users are taken to the dashboard, so they can view information specific to them. Ms Pearcey will also be able to access scheduling functionality after logging in, such as the option to start a new schedule or view an existing one. This section has designed algorithms to meet 1.2 in the structure diagram.

#### 2.5.4 Inputs and validation

Ms Pearcey will first need to submit all inputs to populate the database so that it contains enough data to make the schedule. Whether or not a schedule already exists, there will be a button to start a new schedule.

##### *Start new schedule*

```
## START NEW SCHEDULE
if button clicked
    DROP TABLE Examinees
    DROP TABLE ExamAccomps
    DROP TABLE BasicInfo
    DROP TABLE Schedule
    CREATE TABLE Examinees
    CREATE TABLE ExamAccomps
    CREATE TABLE BasicInfo
    CREATE TABLE Schedule
    redirect to new-schedule.php
endif
```

Figure 2.5.4a: Start new schedule pseudocode

When this button is clicked, the 4 tables shown above are deleted, and created again. This clears all data from the previous exam cycle, so that the current exam cycle can be started. The database is maintained this way, it prevents data being duplicated in these tables, and prevents existing data from affecting future schedules. The user is redirected to

new-schedule.php. This feature has replaced the deletion of this data at 10pm on the last day of exams (see 2.5.1), and the box 1.1.2 in the structure diagram has been updated to reflect this change. Therefore, 1.1.2 in the structure diagram has been designed.

### *Enter initial information*

```
## ENTER START DATE FOR EXAM      | new-schedule.php
if submit button is clicked
    date = get start date and sanitise input
    currentDate = get todays date
    if date=""
        print("You must enter a date.")
    else
        if date < currentDate
            print("The start date must be in the past.")
        else
            Insert date into BasicInfo table
            redirect to accompanist-entry.php
        endif
    endif
endif
```

Figure 2.5.4b: Entering the start date and validation pseudocode

After starting a new schedule, the first thing for users to do is enter the start date. Thanks to HTMLs `<input type="date">`, only a valid date can be submitted. This means that it will already be in the correct format, and most validation is carried out automatically. However, I carry out the validation to check if nothing has been submitted, and give an error message in this case. Also, a user cannot schedule an exam in the past, so the date is validated to be later than (or equal to) the current date. Although email reminders have been removed, I hope to develop this feature later in future iterations of the project. This validation will prevent errors when scheduling emails in the past, making the system more robust.

The date is entered into a new record in BasicInfo. The 2 other fields, IsPublished and Timestamp, are set to False and the current timestamp. This happens automatically, as these are the default values for these fields. The next step is to enter accompanist availability. The Swimlane diagram ([2.2.2](#)) may be quite helpful in this section, as it details the order of functions when scheduling exams (it will contain some features that have since been removed, but is still helpful).

This algorithm has fully designed 1.1.3 in the structure diagram.

## Enter accompanist availability

```
## ENTER ACCOMPANIST INFO | accompanist-entry.php
if submit button is clicked
    email = get email entry
    AccompID = get SupervisorID from Supervisors table for given email
    School = get School entry
    Start0900, Start0930, ... Start1630 = get relevant entries
    if AccompID already in ExamAccomps table
        print("This accompanist's availability have already been entered.")
    else
        if School == true and School == true for any other records in ExamAccomps
            print("Another user has been selected as the school accompanist already.")
        else
            Insert AccompID,School,Start0900,...,Start1630 into ExamAccomps
            print("Successfully added accompanist availability!")
        endif
    endif
endif
```

Figure 2.5.4c: Enter accompanist availability pseudocode

The feature of entering accompanist availability is now just for the organiser. After entering the start date, they will be taken to accompanist-entry.php . At this point, they will be able to add availability for each accompanist manually using a form. This data will populate ExamAccomps.

Ms Pearcey will select an email from a drop down list of registered supervisor emails. This automatically validates the emails, as only valid emails are displayed, so a valid email must be selected. This method ensures that the selected accompanist is already a supervisor, so this does not have to happen manually. The SupervisorID for this user is retrieved, and stored in the variable 'AccompID'.

The fields School,Start0900,...Start1630 are all boolean fields. They will be selected by clicking boxes that represent 30 minute slots in the day, and there will be a final tickbox asking 'Are you the school accompanist?' Once again, the constraints of the input means that only true or false can be submitted for these fields, validating the data type automatically.

Some more validation is carried out to check the data is accurate. If the AccompID is already stored in ExamAccomps, availability has already been entered for the selected supervisor. Therefore, an error message is displayed to prevent the program from trying to enter duplicate AccompIDs. Only one accompanist can be the school accompanist. If School is true, and another accompanist already has School = true, an error message is returned to say 'Another user has been selected as the school accompanist already.' If not, the data is inserted into ExamAccomps. A success message is outputted as well. The user is not redirected anywhere, as Ms Pearcey may want to add availability for more accompanists. After clicking 'submit', the program is run, and a success/error message is displayed. She will then be able to enter availability again, as many times as needed. Once complete, she will click a button called 'Continue to the next stage'. This will take her to examinee-entry.php, and she will submit examinee information here. Structure diagram box 1.4.3 has now been fully designed.

## enter examinee info: examinee-entry.php

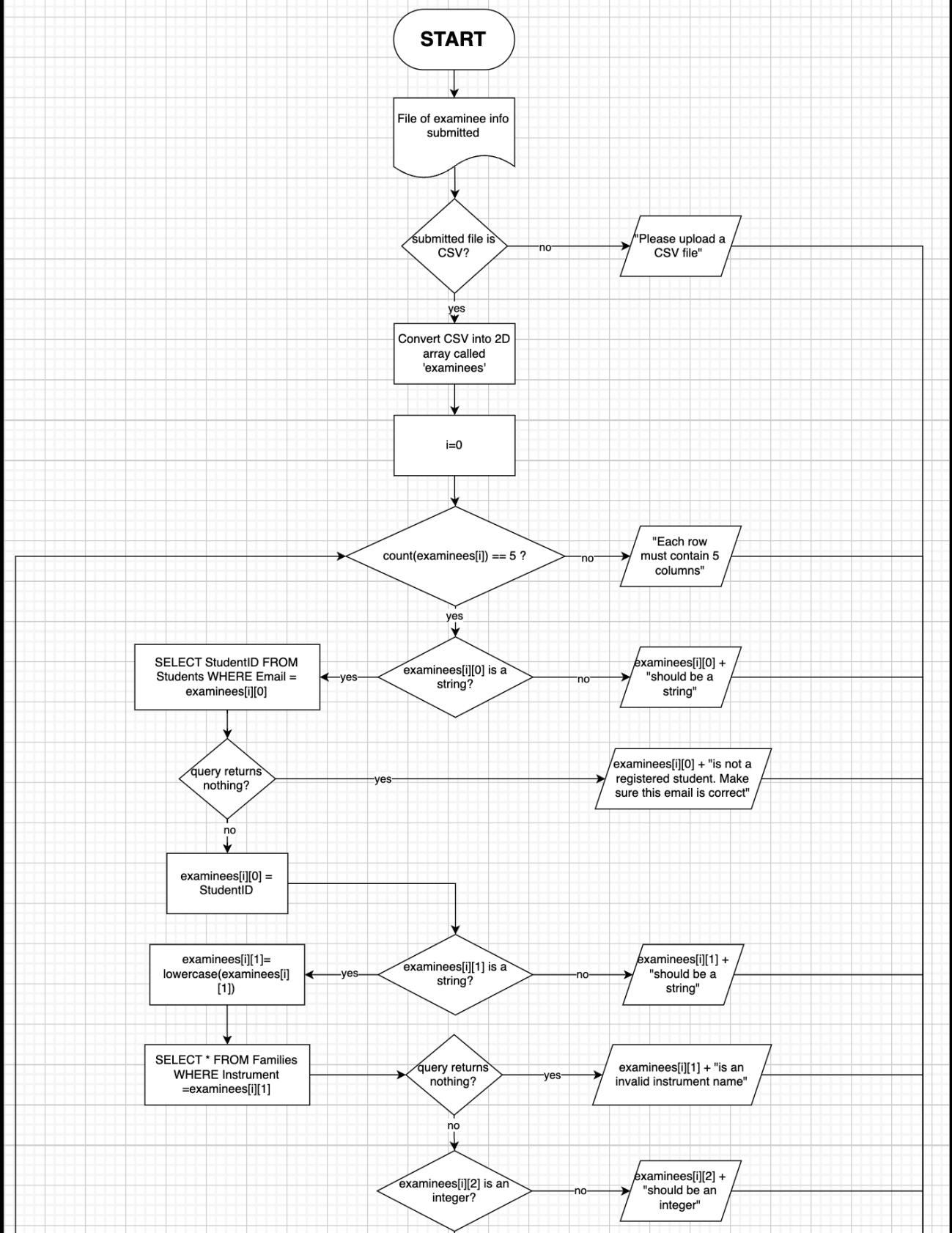


Figure 2.5.4d: Enter examinee info - 1

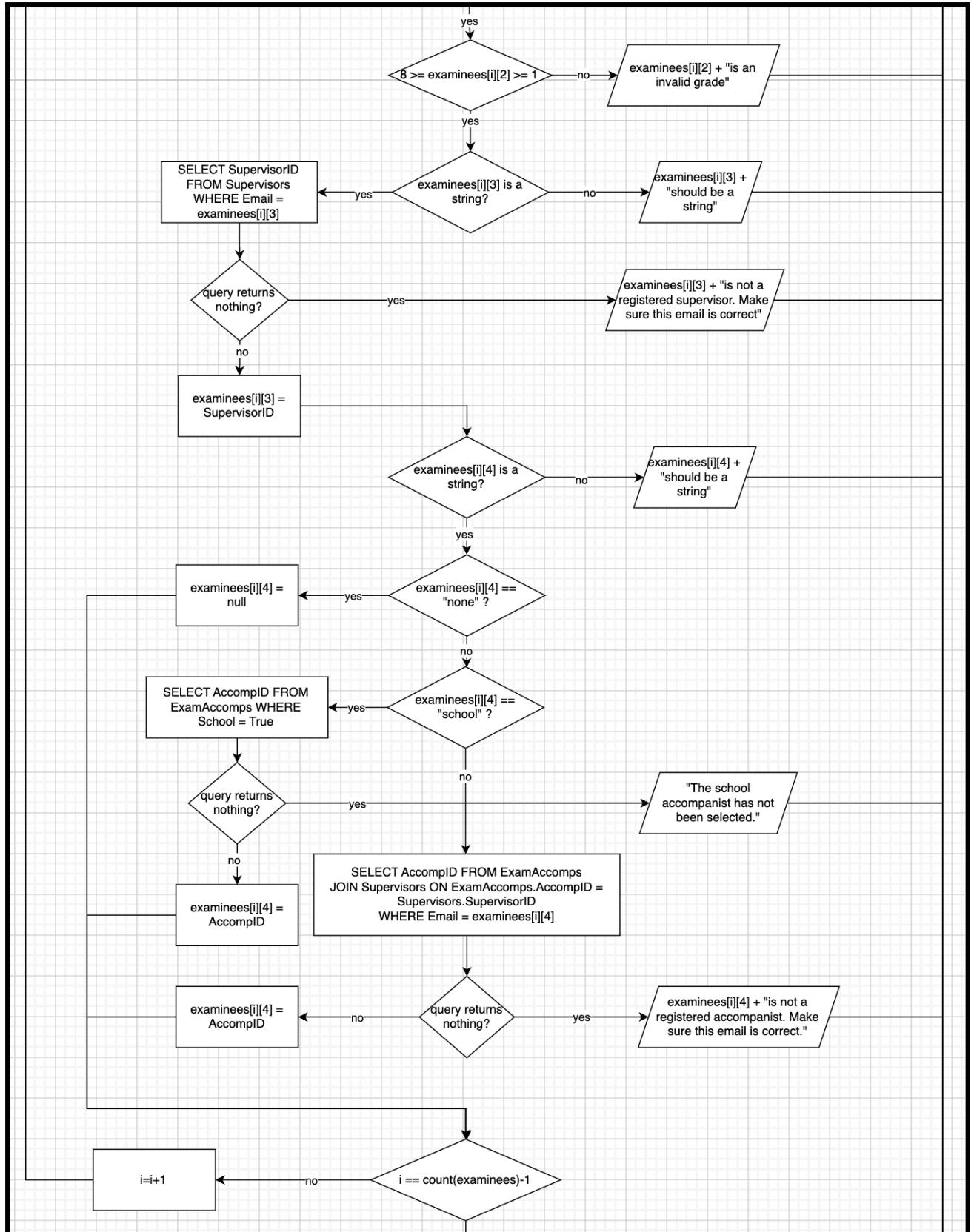


Figure 2.5.4e: Enter examinee info - 2

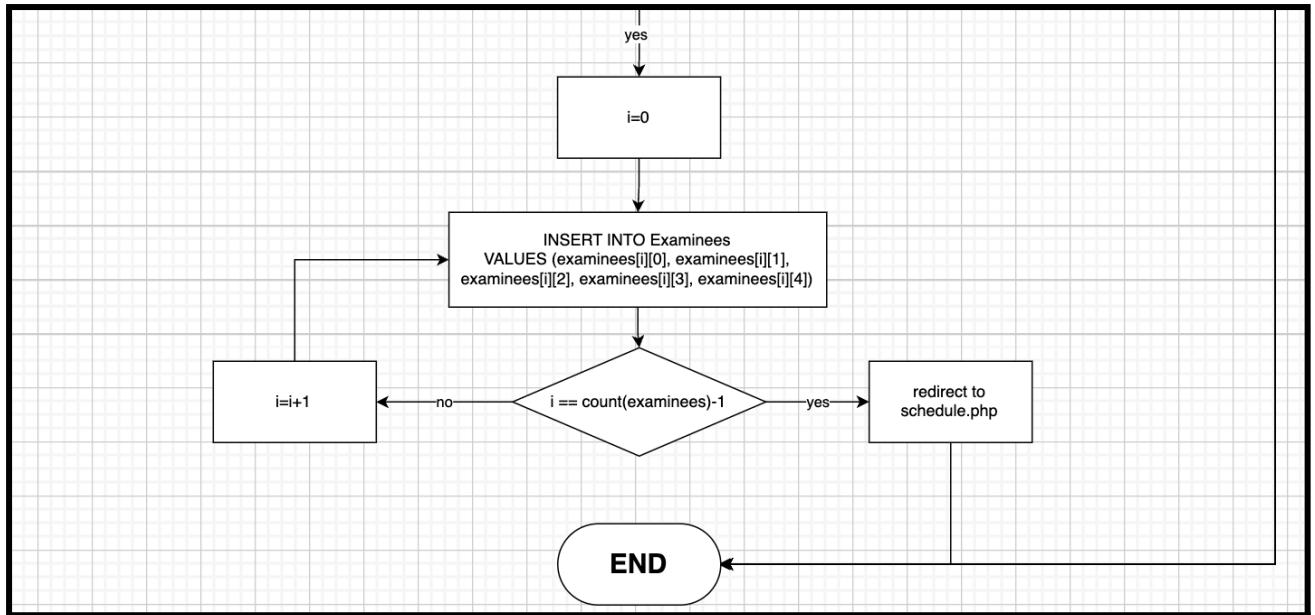


Figure 2.5.4f: Enter examinee info - 3

### *Enter examinee CSV file*

The validation involved in this section is quite lengthy, but all very much necessary. Every single field that is in Examinees is a foreign key for another table, so each field in each row must exist somewhere else in the database. The CSV file is designed to be as easy for Ms Pearcey to create as possible, this has been explained in [2.4](#). Therefore, the file entered will contain several email addresses and other strings, which have to be matched to their relevant IDs. All error messages in this flowchart (and previous sections) are highly specific to the problem, helping the user to correct the mistake easily.

Once the file has been submitted, the program will check if it is a CSV file. If not, an error message is outputted. The CSV is then converted to a 2D array called ‘examinees’. This makes it easier to read and manipulate the data. The CSV file should be in the form shown below:

Field Name	Examinee email address	Instrument	Grade	Music teacher email address	Accompanist email address OR “school” OR “none”
Data type	String	String	Integer	String	String

(repeated) Figure 2.4.1j: format of CSV file for examinees

Therefore, each row in the 2D array should have 5 elements, each one representing the fields shown above (in the same order). Once the array has been created, a large count-controlled loop starts. This will validate each row to have the correct length and data types, check if it is a valid foreign key, and replace some elements with the correct ID. It will also return error messages as necessary. I thought about making this into multiple loops for

each action, but this would involve unnecessary complexity, as one large loop can achieve everything outlined above.

The loop will iterate through each row in the array ‘examinees’. First, it checks if the length of the row is 5. This is checked first to prevent errors with indexing, as a row with 4 elements will not have examinees[i][4], so an error would be returned. Once it has been confirmed to have 5 elements, examinees[i][0] is checked to be a string. This is the examinee’s email address. If it is a string, an SQL query selects the StudentID from Students corresponding to the given email. If nothing is returned, then the given email is not a registered student, or there is a spelling mistake. The email address is returned along with an error message to explain this issue. If something is returned from this query, the value in the array is replaced by the StudentID. This will become the ExamineeID for this examinee (the ExamineeID and StudentID is the same value for each student). This action turns each email address into an ExamineeID, so that the array is closer to the desired output.

examinees[i][index]	0	1	2	3	4
Data stored initially	Examinee email address	Instrument	Grade	Music teacher email address	Accompanist email address OR “school” OR “none”
Data stored finally	ExamineeID (a copy of StudentID)	Instrument (lowercase)	Grade	SupervisorID	AccomplID (or null)

Figure 2.5.4g: How each element in the array should be changed.

The next element to be checked is examinees[i][1], or the instrument. If it is a string, it is converted to lowercase. This improves robustness, as inputs can be in any combination of uppercase and lowercase letters. The instrument is checked if it exists in the Families table by returning all fields where Instrument = examinees[i][1]. In this table, all instruments will be stored in lowercase letters. If nothing is returned, the instrument name is invalid, so an error message is given. If not, then we continue to the next step as the instrument is valid.

examinees[i][2] should be an integer. If it is, then it should exist in the table Timings. I could execute an SQL query to find all fields from Timings where Grade = examinees[i][2], as this would check if the foreign key is valid. However, all music grades are between 1 and 8, so it is simpler to just check this. If they are, we continue to the next element. The valid grade is not changed in the array.

The next element should store a string, containing the music teacher’s email address. If it is a string, the SupervisorID that corresponds to this email is returned from Supervisors. If this ID is returned, it replaces the value of examinees[i][3], to make it more compatible with the database. If not, the email must be incorrectly entered, or the supervisor has not been registered. This error message is displayed, along with the email address causing the error.

The final element in each row of examinees is for accompanists. This should be a string, so this check is carried out. If this string is “none”, then the examinee has no accompanist, so the value of the array is replaced by `null`. If not, the algorithm checks if the string is “school”. In this case, the exam should be accompanied by the school accompanist (ie. the AccomplID where School = True). This accompanist’s AccomplID replaces examinees[i][4]. If no records in ExamAccomps have School set to True, an error message is displayed to say ‘The school accompanist has not been selected.’

If examinees[i][4] is not ‘none’ or ‘school’, it must contain the email address of a registered accompanist. An SQL query returns the AccomplID corresponding to this email address. This query is displayed below:

```
SELECT AccomplID
FROM ExamAccomps
JOIN Supervisors ON ExamAccomps.AccomplID =
Supervisors.SupervisorID
WHERE Email = examinees[i][4]
```

The query cannot simply look through Supervisors, as this table contains all registered music teachers and accompanists. This query must look through the registered accompanists for this exam cycle only, so it looks through ExamAccomps, which is joined with Supervisors to relate AccomplID to email. If this query returns nothing, then there is either a spelling mistake, or the accompanist is not registered for this exam cycle. The error message makes this issue very clear. If an AccomplID is found, examinees[i][4] = AccomplID.

This validation happens for every row of the array. If all validation is passed, then the array should be ready to be inserted into the Examinees table in the database. Originally, I planned to insert this data into the database in the main loop. In this scenario, if the 4th row of the array had an error, then 3 rows would already be in the Examinees table. This is a problem, as it would lead to duplicate entries, or unexpected scheduling results. Therefore, the table is only populated once the **entire** array has been validated.

By looping through each row in examinees, each element is inserted into Examinees as a new record. Due to the formatting carried out in the previous loop, each element in the array is a valid entry into the database, so all relationships connecting to Examinees should be working fine.

At this point in the process: BasicInfo has been populated by entering the start date, ExamAccomps has been populated by entering availability, and now Examinees have been populated by submitting a CSV file. All entries into the database have been carefully validated to ensure that all relationships between tables are strong. Therefore, the actual scheduling process can begin, by redirecting the user to ‘schedule.php’. This is where the schedule function and all of its associated functions will be stored. The schedule function will produce a schedule and display it, along with the options to download a PDF of the schedule and publish the schedule.

This section has fully designed 1.3 in the structure diagram (including 1.3.1, 1.3.2, and 1.3.3).

## 2.5.5 Publishing the schedule

Once the schedule has been produced, it must be displayed and communicated effectively. This relates to box 3 on the structure diagram. Emailing functionality has been removed (see 2.5.1), so 3.2 and 3.4 will not be designed or developed. 3.1 involves displaying relevant information for each user that will be shown when logged in. Everyone will only see the time slots relevant to them, but Ms Pearcey will also be able to view the entire schedule. At this point, she will also be able to download a PDF file of this schedule, so she can print it and stick it on the music notice board.

On the structure diagram, box 3.1.3.1 refers to a function called getRecords that will get the specific data from the database for each user. This data will then be used to display the schedule. This method seemed sensible, but I quickly realised that each user will have to view quite different types of data, and very different SQL queries must be made for each type of user. Each user will also view very specific messages based on the status of the schedule and their user type.

Essentially, the process of viewing a schedule can be split into a few parts:

- Check that something has been published
- If not, display relevant messages
- If so, get the relevant data from the database
- Display the schedule using this data

Displaying relevant messages, and getting relevant information from the database is highly specific to each type of user. This means that it would not make sense to make a function for this, as code will not be repeated if the function didn't exist. However, displaying the schedule using this specific data is quite a general process. No matter who views the schedule, the same method will be used to display the schedule, it will simply use different data. Therefore, I will create a function called displaySchedule to display the schedule, and the function getRecords will not be developed. Its functionality is not being removed, but it is not sensible to make a separate function for such a specific task.

```
if publish button is clicked
    UPDATE basicinfo SET IsPublished=True
    redirect to dashboard.php
endif
```

Figure 2.5.5a: publish button

### *Publish button*

Once the schedule function is finished, it will redirect to 'schedule-complete.php'. Here, the finished schedule will be displayed. This page will also display a button to download a PDF of the schedule, and a button to publish the schedule. Originally, there was going to be a single 'publish' function that would communicate the schedule and download the PDF in one function. However, I have since removed the email functionality and re-evaluated the need

for this function. I think that it would be quite useful for Ms Pearcey to be able to download a PDF **before** publishing the schedule, so I will now include the PDF download as a separate button. As shown above, the publish button will take the record in BasicInfo and make IsPublished = True. This change will cause the schedule to be displayed in specific time slots on everyone's dashboard (dashboard.php).

```
function displaySchedule(result,general)
    for each row in result
        if AccomplID != null
            name = get accompanist name
            result[AccomplID] = name
        else
            result[AccomplID] = ""
        endif
    endfor
    display result
    if general == true
        pdf = new Dompdf
        load result into pdf
        render pdf
        display button to download pdf
    endif
endfunction
```

Figure 2.5.5b: displaySchedule function

*displaySchedule(result,general)*

This function has 2 parameters: result and general. result will be a 2D associative array, containing relevant information which will be displayed, for each time slot that should be displayed. **Each row in result will contain the examinee's first name, last name, instrument, grade, AccomplID, and start time.** The SQL query made to obtain this result will differ each time this function is called. result may contain 1 row for one student, multiple rows for a supervisor, or all of the rows for the admin.

In order to view the full existing schedule, schedule-complete.php will call this function with general = True. As the general schedule is being displayed, the button for downloading the PDF should also be displayed. However, dashboard.php will call this function with general =

False. In this case, a specific schedule is being displayed, so there is no need to give users access to the PDF.

The AccomplID is replaced with the accompanist's name, by making an SQL query to obtain it. If AccomplID is null, it is replaced with an empty string to maintain the format of the table. Then, the array is displayed. It will be displayed in a table format.

The DOMPDF library will be used to create this pdf. By starting a new Dompdf, a new instance of this class will be created. The array will then be loaded into the pdf. This step will involve converting each row and element into html and php that will display the table in an easy to understand, user-friendly format. It will then be rendered, and a button to download this file will be displayed on screen. I chose DOMPDF as it is very good at converting HTML into a PDF format, giving me more control over how the PDF should be styled and displayed. I was also considering the TCPDF library due to its customizability features, but I learned that it did not support HTML as well as DOMPDF, making it the worse option.

This function allows existing schedules to be loaded and displayed, so 1.1.1 in the structure diagram has been designed. It has also designed 3.3 in the structure diagram, as this function will display a button to download the PDF file of the schedule. The next section will call displaySchedule for 3.1 in the structure diagram to be designed fully.

```
## DISPLAY SPECIFIC AND RELEVANT INFO | dashboard.php
results_message='If you have recently taken an exam, results will be displayed
on the music notice board, or you will be contacted by your music teacher.'
isPublished=SELECT IsPublished FROM BasicInfo
if isPublished == true
    if user_type == 'student'
        result = get relevant fields from database for given ExamineeID
        if result is empty
            print('You arent scheduled to play in the upcoming exams.')
            print(results_message)
        else
            displaySchedule(result,false)
        endif
    else
        result = get relevant fields from database for given ID as a teacher
        if result is empty
            print('You arent teaching any upcoming examinees.')
        else
            displaySchedule(result,false)
        endif
        result = get relevant fields from database for given ID as an accompanist
        if result is empty
            print('You arent accompanying any upcoming exams.')
        else
            displaySchedule(result,false)
        endif
    endif
else
    print('Nothing has been scheduled yet. Check again later.')
    if user_type=='student'
        print(results_message)
    endif
endif
print('Email Ms. Pearcey at admin@email.com if you have any concerns.')
```

Figure 2.5.5c: displaying specific information on the dashboard

### *Displaying specific and relevant information*

This part of the program will be developed in dashboard.php. It will allow success criteria L7-10 to be achieved, which includes the entirety of structure diagram box 3.1. This will decide what should be displayed on the dashboard for each user. Firstly, it will check the value of IsPublished, which is a field in BasicInfo. By default, this is false, but the publish button sets it to true. It is also possible that BasicInfo is empty. In this case the first if statement would be false. If nothing has been scheduled, or nothing has been published, everyone will see a message saying ‘Nothing has been scheduled yet. Check again later.’ As specified in my success criteria, students will receive an additional message to inform them about how results are communicated to them. This prevents any confusion for examinees who have recently taken exams by clarifying the situation.

If the schedule has been published, the session variable user\_type is checked. If a student has logged in, user\_type=’student’ so an SQL query will use their StudentID to retrieve the time slot relevant to them, and its associated data. **Each row in result will contain the examinee’s first name, last name, instrument, grade, AccomplID, and start time.** This data will be stored ‘result’. If the student is not playing in any exams, they will not appear in the schedule, so ‘result’ will be empty. If so, they will receive a message to say ‘You aren’t scheduled to play in the upcoming exams’, as well as the results message. If ‘result’ is not empty, it is passed as a parameter when displaySchedule is called. This will cause the student to see when their exam is. Please note that the session variable user\_id will allow the program to access the user’s StudentID/SupervisorID.

If the user\_type is not ‘student’, it is ‘supervisor’ or ‘admin’. As Ms Pearcey will also want to view any exams she is involved in as a teacher / accompanist, both ‘supervisor’ and ‘admin’ should result in the same instructions being carried out. Therefore, if the user type is not a student, an SQL query will use their SupervisorID to get all of the time slots for students they are teaching. This query will directly join Supervisors to Examinees. If the result is empty, ‘You aren’t teaching any upcoming examinees.’ is displayed. If not, displaySchedule is called, and the result is passed as a parameter. Additionally, another SQL query will use the user\_id as an AccomplID to get all time slots for exams they are accompanying. This is stored in ‘result’. Again, if this result is empty, this person is not accompanying anyone soon, so an appropriate message is displayed. If not, displayedSchedule is called, passing ‘result’ as a parameter. Each time this function is called, ‘result’ will store vastly different groups of the schedule. This function displaySchedule makes the system more modular and consistent, and prevents repeated code. In this section, the parameter ‘general’ = false. General is only set to true when Ms Pearcey views the schedule in schedule-complete.php.

No matter what, there will be a message at the bottom of the page telling users to email Ms Pearcey if there are any concerns. These helpful messages make the system more usable, but have all been carefully thought out to promote certain actions. For example, the message to email Ms Pearcey has been designed after a survey of recent examinees revealed that some people had an issue with their time slot, but said nothing. This message should promote communication between Ms Pearcey and students. This message, along with

all messages, will meet success criteria L7-10. Structure diagram box 3.1 has now been fully designed.

## 2.5.6 Meeting the success criteria

The algorithms designed above can together be developed to create a complete solution, as defined in section 2.2 (and re-defined in 2.5.1). The table below displays which parts of the structure diagram, and which algorithms link to the success criteria. By developing every algorithm, all of the success criteria can be met to achieve a successful, complete project. Please note that features removed in 2.5.1 do not have an associated algorithm, as they have been removed from the solution. These rows will be highlighted in red. All other success criteria have a matching part of the structure diagram and a matching algorithm. Success criteria G1 has been removed from the table, as it is met by developing a simple and intuitive interface. In order to meet G1, I must develop the usability features outlined in section [2.3](#), its success is not tied to anything in the structure diagram. This is the only success criteria that has been removed from the table.

The success criteria have been referenced in the same style as section 1.7. The structure diagram has been labelled clearly in section 2.2.1, these labels have been used below. The algorithms have been referenced by the section they are in (eg. 2.5.2 means an algorithm designed in section 2.5.2).

Success Criteria Ref.	Structure diagram	Algorithm
L1	1.2	2.5.3
L2	3.1	2.5.5
L3	1.2.1.1	2.5.3
L4	1.2.1.1	2.5.3
L5	1.2.1.3	
L6	1.2.1.2	
L7	3.1.3.1	2.5.5
L8	3.1.1.2	2.5.5
L9	3.1.2	2.5.5
L10	3.1.1.1	2.5.5
E1	1.1.1	2.5.5
E2	1.1.2 (this box was updated in section 2.5.1 in the document)	2.5.4
E3	2.3	

I1	1.1.3	2.5.4
I2	1.3.1	2.5.4
I3	1.3.2/3	2.5.4
I4	1.4.3	2.5.4
I5	1.4.1	
I6	1.4.2	
I7	1.4.1.1	
I8	1.4.4	
S1	2.1.3	2.5.2
S2	2.1.1.1	2.5.2
S3	2.1.4	2.5.2
S4	2.1.1.1	2.5.2
S5	2.1.1	2.5.2
S6	2.1.2	2.5.2
S7	2.1.1.1	2.5.2
S8	2.1.2.1	2.5.2
S9	2.2.1	
S10	2.2.2	
S11	2.2.3	
P1	3.2	
P2	3.2.1/2	
P3	3.4	
P4	3.3	2.5.5
P5	3.3.1/2	2.5.5

Figure 2.5.6a: Showing how all designed algorithms form a complete solution to the project

In order to reach a complete solution, all success criteria must be met (except those removed in 2.5.1). As this table shows, all success criteria can be met by implementing my structure diagram and, more importantly, developing every algorithm shown in section [2.5](#). A successful and complete project can clearly be achieved by implementing all algorithms shown in this section.

## 2.6 Iterative test data

### 2.6.1 Unit and system testing

Iterative testing is a key aspect of the iterative development process, as it allows me to identify if all modules can function independently, and identify if they can work together. My development will be divided into stages, and each stage will involve iterative testing. The testing will allow me to make any necessary changes to the code until the module passes all tests. I will repeat this iterative development and testing process for each stage. This is known as unit testing, as I am testing the complete functionality of individual modules. Unit testing is crucial, because inputs can be controlled to test the full range of each module's functionality with valid, boundary and invalid/erroneous data. I will also conduct system testing during my development. After several modules have been developed and carefully tested, they will be integrated into a larger system. Testing this system allows me to evaluate how the modules work together and gain a deeper insight into their accuracy. This will be conducted using real data provided by Ms Pearcey. Each iterative stage will create a prototype, allowing me to move closer to my final solution.

### 2.6.2 Real exam data

Lorne Pearcey shared a spreadsheet

Lorne Pearcey ([l.pearcey@camphillboys.bham.sch.uk](mailto:l.pearcey@camphillboys.bham.sch.uk)) has invited you to edit the following spreadsheet:

Here is the data for your project. N.B. It is not anonymised so please be sensible with its use, or anonymise it yourself!

**TCL data for Rohan**

Candidate's first name	Candidate's last name	Instrument	Grade	Teacher	Exam length	Accompanist
Aishwarya	Siddharth	Classical Guitar	Grade 1	Mr Phillips	00:13 -	
Aisha	Naveena	Classical Guitar	Grade 1	Mr Phillips	00:13 -	
Aishwarya	Rajesh	Trumpet	Grade 1	Mr Phillips	00:13 ED	
Aisha	Reyanah	Piano	Grade 4	Mr Hey	00:23 GFP	
Aishwarya	Shreyas	Classical Guitar	Grade 1	Mr Phillips	00:13 -	
Aishwarya	Kavita	Classical Guitar	Grade 3	Mr Phillips	00:13 -	
Aishwarya	Hannah	Classical Guitar	Grade 1	Mr Phillips	00:13 -	
Aishwarya	Danielle	Classical Guitar	Grade 2	Mr Rose	00:13 -	
Aishwarya	Divya	Trumpet	Grade 1	Mr Drew	00:13 ED	
Aishwarya	Divya	Classical Guitar	Grade 1	Mr Phillips	00:13 -	
Esha	Divyanshu	Classical Guitar	Grade 1	Mr Phillips	00:13 -	
Fawzia	Divyanshu	Classical Guitar	Grade 1	Mr Phillips	00:13 -	
Harsh	Mihir	Drums	Grade 1	Mr Phillips	00:13 -	
Jay (Roh)	Chait	Clarinet	Grade 5	Mr Mehta	00:13 JM	
Jay (Roh)	Chait	Drums	Grade 1	Mr Phillips	00:13 ED	
Liyah	Rishabh	Trumpet	Grade 5	Mr Butler	00:13 GFP	
Utkarsh	Leilani	Vocal	Grade 4	Mr Phillips	00:13 GFP	
Utkarsh	Leilani	Vocal	Grade 4	Mr Phillips	00:13 GFP	

Lorne Pearcey is the owner

Last edited by Lorne Pearcey 5 days ago

[Open](#)

Figure 2.6.1a: Ms Pearcey sharing TCL data

	A	B	C	D	E	F	G
1	Candidate's first name	Candidate's last name	Instrument	Grade	Teacher	Exam length	Accompanist
2	Alicia	Dieu	Classical Guitar	Grade 3	Mr Phillips	00:13	-
3	Ananya	Tare	Classical Guitar	Grade 3	Mr Phillips	00:13	-
4	Anupama	Harish	Violin	Grade 6	Ms Millar	00:23	SFP
5	Anushtup	Chatterjee	Violin	Grade 6	Mr Georgiev	00:23	SFP
6	Basheir	Said	Trumpet	Grade 1	Mrs Butler	00:13	SFP
7	Benjamin	Law	Trumpet	Grade 6	Mrs Butler	00:23	SFP
8	Charlotte	Wood	Clarinet	Grade 7	Mr Meadows	00:23	JM
9	Clover	Webster	Cornet	Grade 5	Mrs Butler	00:18	SFP
10	Daniel	Okpla	Classical Guitar	Grade 1	Mr Rose	00:13	-
11	Edward	Cheung	French Horn	Grade 2	Mrs Butler	00:13	SFP
12	Ella	Goldberg	Clarinet	Grade 6	Mr Meadows	00:23	JM
13	Kirsty	Jenkins	Clarinet	Grade 7	Mr Meadows	00:23	JM
14	Lemuel	Adjei	Trumpet	Grade 3	Mrs Butler	00:13	SFP
15	Levi	Dandy	Classical Guitar	Grade 3	Mr Rose	00:13	-
16	Maryam	Mohamed	Clarinet	Grade 5	Mr Meadows	00:18	JM
17	Matteo	Gianni	Baritone	Grade 6	Mr Hickman	00:23	SFP
18	Olivia	Yang	Classical Guitar	Grade 2	Mr Phillips	00:13	-
19	Reena	Koiri	Classical Guitar	Grade 2	Mr Phillips	00:13	-
20	Sam	Nouhov	Jazz Saxophone	Grade 6	Mr Drew	00:23	ED
21	Sana	Khan	Classical Guitar	Grade 3	Mr Phillips	00:13	-
22	Shreya	Jondhale	Violin	Grade 5	Ms Millar	00:18	SFP
23	William	Lin	Violin	Grade 5	Mr Georgiev	00:18	SFP
24							
25							
26					Initial	00:11	
27					Grade 1	00:13	
28					Grade 2	00:13	
29					Grade 3	00:13	
30					Grade 4	00:18	
31					Grade 5	00:18	
32					Grade 6	00:23	
33					Grade 7	00:23	
34					Grade 8	00:28	
35							
36					SFP	After 11:00	
37					ED	Before 11:30	
38					JM	10:00 - 3:30	
39							

+   ≡    Winter 2023 ▾    Summer 2023 ▾    Spring 2023 ▾

Figure 2.6.1b: Real exam data - Spring 2023

	A	B	C	D	E	F	G
1	Candidate's first name	Candidate's last name	Instrument	Grade	Teacher	Exam length	Accompanist
2	Aakshat	Kumar	Classical Guitar	Grade 3	Miss Palmer	00:13	-
3	Aman	Koiri	Clarinet	Grade 5	Mr Meadows	00:18	JM
4	Anesha	Mitra	Classical Guitar	Grade 5	Mr Phillips	00:18	-
5	Birle	Tenekeci	Violin	Grade 2	Mr McGee	00:13	SFP
6	Catherine Mae	Villabroza	Jazz Saxophone	Grade 7	Mr Drew	00:23	ED
7	Chardé	Levermore	Classical Guitar	Grade 3	Mr Phillips	00:13	-
8	Clara	Hilton-Widdows	Clarinet	Grade 5	Mr Meadows	00:18	JM
9	Deeksha	Sharma	Trombone	Grade 4	Mrs Butler	00:18	SFP
10	Diya	Sharath	Classical Guitar	Grade 1	Mr Phillips	00:13	-
11	Elizabeth	Shpectorov	Classical Guitar	Grade 7	Mr Phillips	00:23	-
12	Inaaya	Kassim	Violin	Grade 1	Mr McGee	00:13	SFP
13	Louisa	Dilling	Violin	Grade 6	Mr McGee	00:23	SFP
14	Lucas	Evans	Classical Guitar	Grade 1	Miss Palmer	00:13	-
15	Marriam	Javed	Trombone	Grade 1	Mrs Butler	00:13	SFP
16	Mikayla	Rodney	Saxophone	Grade 3	Mr Drew	00:13	ED
17	Mithun	Kesavan	Classical Guitar	Grade 1	Miss Palmer	00:13	-
18	Rinsola	Alatise	Clarinet	Grade 5	Mr Meadows	00:18	JM
19	Ryheem	Miah	Saxophone	Grade 1	Mr Drew	00:13	ED
20	Samia	Islam	Clarinet	Grade 5	Mr Meadows	00:18	JM
21	Sana	Khan	Classical Guitar	Grade 4	Mr Phillips	00:18	-
22	Sara	Sadiq	Classical Guitar	Grade 1	Mr Phillips	00:13	-
23	Tamsin	Howard	Violin	Grade 7	Mr McGee	00:23	SFP
24	Zaynah	Jamal	Classical Guitar	Grade 2	Mr Phillips	00:13	-
25	Zubair	Ahmed	Classical Guitar	Grade 1	Miss Palmer	00:13	-
26							
27							
28					Initial	00:11	
29					Grade 1	00:13	
30					Grade 2	00:13	
31					Grade 3	00:13	
32					Grade 4	00:18	
33					Grade 5	00:18	
34					Grade 6	00:23	
35					Grade 7	00:23	
36					Grade 8	00:28	
37							
38					SFP	Anytime	
39					ED	After 12:45	
40					JM	9:15 to 14:15	

+   ≡    Winter 2023 ▾
Summer 2023 ▾
Spring 2023 ▾

Figure 2.6.1c: Real exam data - Summer 2023

	A	B	C	D	E	F	G
1	<b>Candidate's first name</b>	<b>Candidate's last name</b>	<b>Instrument</b>	<b>Grade</b>	<b>Teacher</b>	<b>Exam length</b>	<b>Accompanist</b>
2	Aakshat	Kumar	Classical Guitar	Grade 4	Miss Palmer	00:18	-
3	Ahana Aditi	Seeam	Classical Guitar	Grade 1	Mr Phillips	00:13	-
4	Aisha	Nashmia	Classical Guitar	Grade 1	Mr Phillips	00:13	-
5	Akshay	Suglani	Saxophone	Grade 3	Mr Drew	00:13	ED
6	Aliza	Rayhan	Flute	Grade 6	Mr Hay	00:23	SFP
7	Ananya	Tare	Classical Guitar	Grade 4	Mr Phillips	00:18	-
8	Anisha	Kawle	Classical Guitar	Grade 3	Mr Phillips	00:13	-
9	Aroush	Haider	Classical Guitar	Grade 1	Mr Phillips	00:13	-
10	Daniel	Okpla	Classical Guitar	Grade 2	Mr Rose	00:13	-
11	Doyinsola	Oliyide	Saxophone	Grade 1	Mr Drew	00:13	ED
12	Eshal	Aamir	Classical Guitar	Grade 1	Mr Phillips	00:13	-
13	Farishta	Dosanjh	Classical Guitar	Grade 1	Mr Phillips	00:13	-
14	Henna	Naveed	Clarinet	Grade 5	Mr Meadows	00:18	JM
15	Jiayi (Eva)	Chen	Clarinet	Grade 3	Mr Meadows	00:13	JM
16	Lauren	Pumphrey	Flute	Grade 6	Mr Hay	00:23	SFP
17	Leya	Rahman	Trumpet	Grade 5	Mrs Butler	00:18	SFP
18	Lily-Marie	Le Blanc	Viola	Grade 3	Ms Keum	00:13	SFP
19	Maryam	Rahman	Violin	Grade 5	Mr McGee	00:18	SFP
20	Milaaura	Fernando	Violin	Grade 1	Mr McGee	00:13	SFP
21	Nikhil	Gilliam	Classical Guitar	Grade 4	Miss Palmer	00:18	-
22	Shailey	Sivathasan	Classical Guitar	Grade 1	Mr Phillips	00:13	-
23	Sri	Grandhi	Classical Guitar	Grade 3	Mr Rose	00:13	-
24	Tula	Hobbs	Classical Guitar	Grade 5	Mr Phillips	00:18	-
25	Xander	Davis	Trumpet	Grade 6	Mrs Butler	00:23	SFP
26	Zaid	Rassam	Trumpet	Grade 1	Mrs Butler	00:13	SFP
27							
28							
29					Initial		00:11
30					Grade 1		00:13
31					Grade 2		00:13
32					Grade 3		00:13
33					Grade 4		00:18
34					Grade 5		00:18
35					Grade 6		00:23
36					Grade 7		00:23
37					Grade 8		00:28
38							
39					SFP	Anytime	
40					ED	Before 11:30	
41					JM	Between 9:30 and 3:30	

+

☰ Winter 2023 ▾

Summer 2023 ▾

Spring 2023 ▾

Figure 2.6.1d: Real exam data - Winter 2023

This real data has been provided by Ms Pearcey. It contains examinee and accompanist information for 3 exam cycles. Each one can be used to populate the database and create a schedule, which can then be compared to the schedules created by Ms Pearcey for these exams. I aim to use 1 of the above sets of data for iterative testing purposes, as it can be used to help me adjust code and correct errors. As this data is real, meticulous system tests will provide a deep understanding into how the software will perform in real life. Any errors found using this data can then be corrected, making my system more effective and robust. I also aim to get stakeholder feedback during these system tests, implementing any requested changes to allow my project to be as useful as possible. I will leave 2 sets of data for

post-development testing, which will allow me to make an accurate assessment of the success of the project during my evaluation.

### 2.6.3 Types of test data

The data supplied by Ms Pearcey provides a great example of valid test data. However, my iterative testing must test the full functionality of each module, especially during unit testing at the end of each stage. This means that boundary data, and invalid/errorneous data must be entered into the system. Therefore, I will also generate a set of fake data to populate the database. This can be manipulated by me in any way, which gives me full control over what is inputted into the system for each test. This is how I will be able to generate the majority of valid, invalid, and boundary data during iterative testing, so that each module and prototype can be comprehensively tested. This will test the robustness of the system, as well as its functionality and usability.

For each module, I have given a list of some tests that need to be carried out during iterative testing.

#### *Login and registration*

- Can students be registered separately to supervisors?
- Presence check for all data prevents nothing from being entered
- Invalid names, emails, and passwords are detected, with an error message displayed
  - Invalid names: '123', 'test£'
  - Invalid email: 'a@b'
  - Invalid password: 'password1', 'Password', 'Pas1'
- Is the hashed password stored in the database?
- Can a correct password log the user in?

#### *Gathering inputs*

- Is an invalid start date rejected? (eg. 07/08/2002 is in the past so should be rejected)
- Is the examinee file rejected if not a CSV?
- Is every element in the file carefully validated against the database?
  - Invalid data eg. grade: '10', or instrument name: 'music' is rejected
- Can only registered supervisors be selected as accompanists?

#### *Scheduling exams*

- Given valid data in the database, does the produced schedule meet TCL regulations?
- Does the schedule recognise when too many examinees are entered into the database, and does it end the day appropriately in this case?
- Does the schedule recognise when insufficient availability is entered into the database, and does it return an error message?
- Does the schedule try to schedule breaks at the times specified by Ms Pearcey in section [1.1.3](#)?
- Does the software sort exams by accompanist free time?
- Does it sort exams by instrument family, then instrument, then grade?
- Does the schedule always adhere to accompanist availability?

### *Publishing results*

- When the schedule is displayed, are all relevant fields included in an easy-to-understand interface?
- Are relevant messages displayed on the user dashboard, across many possible scenarios?
- Is the data displayed specific and relevant, with unnecessary data abstracted?
- Does the publish button change all user dashboards to their appropriate time slots?
- Can the PDF be downloaded?
- Does the PDF display the full schedule?

### *Usability features*

- Do all buttons and links function correctly?
- Is the interface simple and clear?
- Are error messages displayed clearly in a red colour?
- Are error messages specific and directly focussed on the issue?
- Are helpful messages displayed throughout the process?
- Is the interface consistent across pages?

## 2.7 Post-development test data

Once development is complete, the system must be tested to assess the success of the final solution. This will include testing the functionality of each module, testing the robustness of the system, and testing its usability. Stakeholder feedback will be crucial in this section, especially Ms Pearcey's feedback. I plan to conduct a detailed interview with her after development is complete, where she can provide honest feedback about each aspect of the solution. I will also contact Cyrus and Darius for feedback about the usability of the system.

In order to test the functionality of the entire system, I will use 2 sets of the real data provided by Ms Pearcey. This data will be used to register students, and to log them in. It will be used to create a CSV file of examinees, which will then be submitted into the website. It will be used to enter accompanist availability into the website, and therefore a schedule of this real data can be produced. This can be used to assess the functionality of the entire system, but especially the scheduling aspect. I will compare the produced schedule to the one produced by Ms Pearcey in my evaluation to clearly see how my algorithm performs in real conditions.

I have designed a number of tests to be carried out during my post-development testing.

These are split into:

- Functionality tests (which are split further by module)
- Robustness tests
- Usability tests

Robustness tests will always involve rejected inputs, ie. displaying an error message. The functionality tests will involve processing accepted inputs. Tests are numbered, and named by the category they are in (eg. Test L1). This slightly mimics the way success criteria are named, to reinforce the link between tests and the success criteria they are testing. I have not designed any tests to meet success criteria that were removed in section [2.5.1](#). The table in section [2.5.6](#) clearly illustrates which criteria have been removed.

### 2.7.1 Post-development functionality testing

#### (L) Login and registration

L	Description	Success Criteria met
1	Students can log in correctly, taken to the dashboard	L1
2	Supervisors can log in correctly, taken to the dashboard	L1
3	Admin can log in correctly and is taken to the dashboard	L1
4	Students can register themselves correctly, and are redirected to the login page	L3
5	Supervisors can register themselves correctly, and are redirected to	L3

	the login page	
6	Clicking logout will log the user out, taking them back to the index page	L1
7	Only admin can access scheduling functionality, as all pages should redirect unauthorised users away.	L2
8	The dashboard includes a link to the admin dashboard for the admin only	L2

(I) *Gathering inputs for a new schedule*

I	Description	Success Criteria met
1	Start new schedule button will delete and create BasicInfo, Examinees, ExamAccomps and Schedule in the database (to prepare the database for a new schedule) and redirect to new-schedule.php	E2
2	The exam's start date (valid data) can be entered and is stored in the database	I1
3	Valid accompanist availability can be entered and is stored in the database	I4
4	A valid CSV file of examinee information can be entered and is stored in the database	I2

(S) *Scheduling*

S	Description	Success Criteria met
1	The schedule function will produce a schedule that meets all TCL regulations	S1
2	Breaks are scheduled within 10 minutes of the ideal break times set by Ms Pearcey	S1
3	The scheduled wastes as little time as possible	S2
4	The most urgent exams are attempted to be scheduled first	S2
5	The schedule always adheres to accompanist availability	S6
6	Exams are reasonably grouped by accompanist	S4
7	Within each accompanist block, exams are grouped by instrument family	S5

8	Within each instrument family, exams are grouped by instrument	S5
9	Within each instrument, exams are sorted ascending by grade	S5
10	The schedule is stored within the Schedule table in the database	S8

### (P) Publishing

P	Description	Success Criteria met
1	'View Existing Schedule' button takes user to schedule-complete.php and displays the full schedule	E1
2	'Publish' button will set IsPublished to 1 and redirect the user to the dashboard	L7
3	If nothing is published, users will receive a message to acknowledge this on their dashboard.	L8
4	If nothing is published, students will receive an additional message regarding the communication of their results	L10
5	Once published, all students view specific information about their upcoming exam	L7
6	Once published, all supervisors view specific schedule information about upcoming exams they are accompanying	L7
7	Once published, all supervisors view specific schedule information about upcoming examinees they are teaching.	L7
8	Each user's timetable abstracts any unnecessary data from view	L7
9	All users will see a message to contact Ms Pearcey if there are any concerns, with her email included in this message.	L9
10	The admin is able to download a PDF of the schedule by clicking the 'Download PDF' button	P5
11	The PDF contains the full schedule, with the following fields displayed: Examinee name, Instrument, Grade, Accompanist name, Start time	P4

### 2.7.2 (R) Post-development robustness testing

Data used in these tests is invalid data

R	Description	Success Criteria met

1	Empty fields are rejected during registration	L4
2	Invalid name is rejected during registration	L4
3	Invalid email is rejected during registration	L4
4	Invalid password is rejected during registration	L4
5	A registered email is rejected during registration	L4
6	Unregistered (invalid) email is rejected during login	L1
7	Invalid password is rejected during login	L1
8	If the date is in the past, it is rejected.	I1
9	There is a drop down list that only displays registered supervisors, so only they can be selected as accompanists	I4
10	If one user is already the school accompanist, selecting the school accompanist for another user will return an error message.	I4
11	If the examinee file is not a CSV, it is rejected.	I3
12	When entering examinee info, if each row does not contain 5 elements with correct data types, the file is rejected.	I3
13	When entering examinee info, if an element is not verified as a valid foreign key in the database, the entire file is rejected.	I3
14	The schedule is constantly validated to be meeting the regulations, and if an error is found, the program stops and displays it.	S1
15	When too many examinees are entered into the database, the day is ended before 5pm to ensure a valid schedule is produced	S1
16	Only once regulations are broken will the schedule end, in order to schedule as many exams as allowed.	S3
17	When insufficient availability is entered, the function will recognise this and display an error message.	S7
18	The error message will display the specific names of accompanists with insufficient availability	S7

### 2.7.3 Post-development usability testing

To accurately test the usability of the system, I will be asking the following questions to all stakeholders:

1. Do you think this interface is simple and intuitive?
2. Are the scheduling results easy to understand?
3. Would you want any other data to be displayed?
4. Is the on-screen help useful?

5. Is there any unnecessary information on screen at any point in the system?
6. Would you change anything to make the system more usable?

I will ask these further questions to Ms Pearcey:

7. Is the 'schedule' button easy to press?
8. Is the 'publish' button easy to press?
9. Considering people with varying age and technological dexterity will use this system, would you say that the interface is easy to understand?

Getting feedback from Cyrus, Darius, and Ms Pearcey will allow me to fully understand how each user interacts with the system, and how comfortable they feel with it. Ms Pearcey's feedback will be the most important, as she will be using this system the most, and will use the full range of its functionality.

# 3 Development

My development has been split into appropriate stages, based on the different modules of my solution. Please note: my design has changed during my development. I have specified these changes at the end of each stage of development.

## 3.1 Database setup

### 3.1.1 Developing and testing

In my design, I stated that some tables would be created by the program at the start of each cycle, and deleted at the end. These tables are: Examinees, ExamAccomps, BasicInfo and Schedule. The deletion of these tables would be executed using the DROP command, eg. `DROP TABLE Examinees;`. However, I now understand that this is not the best choice. The TRUNCATE command is more efficient, as it deletes all records **without** deleting the table. This eliminates the need to create the tables repeatedly, as each exam cycle will start with an empty table. Therefore, at the end of each exam cycle, I will execute this command, eg. `TRUNCATE TABLE Examinees;`. This makes the code simpler and improves performance.

For this reason, I am creating all tables at the start of my development, and none will be deleted. Only the tables Families and Timings will be populated here, as they contain static data. I am using phpMyAdmin with MySQL to manage my database.

Firstly, I decided to create Timings and Families. Using phpMyAdmin's interface, I could specify the data types and other attributes. However, foreign keys can only be added manually using SQL. This is shown when creating the foreign key in Families.

The screenshot shows the phpMyAdmin interface for creating a new table named 'Timings'. The 'Structure' tab is active. The table has three columns: 'Grade' (INT), 'InstrumentFamily' (VARCHAR(191)), and 'Duration' (INT). Both 'Grade' and 'InstrumentFamily' are marked as PRIMARY keys. The 'Storage Engine' is set to MyISAM. At the bottom, there are 'Preview SQL' and 'Save' buttons.

Creating Timings table using phpMyAdmin

#	Name	Type	Collation	Attributes	Null	Default	Comments	Extra	Action
1	Grade	int			No	None			<a href="#">Change</a> <a href="#">Drop</a> <a href="#">More</a>
2	InstrumentFamily	varchar(191)	utf8mb4_0900_ai_ci		No	None			<a href="#">Change</a> <a href="#">Drop</a> <a href="#">More</a>
3	Duration	int			No	None			<a href="#">Change</a> <a href="#">Drop</a> <a href="#">More</a>

Timings table with all attributes

#	Name	Type	Collation	Attributes	Null	Default	Comments	Extra	Action
1	Instrument	varchar(191)	utf8mb4_0900_ai_ci		No	None			<a href="#">Change</a> <a href="#">Drop</a> <a href="#">More</a>
2	InstrumentFamily	varchar(191)	utf8mb4_0900_ai_ci		No	None			<a href="#">Change</a> <a href="#">Drop</a> <a href="#">More</a>

Families table has been created

```

1 ALTER TABLE families
2 ADD CONSTRAINT families_fk_1
3 FOREIGN KEY (InstrumentFamily)
4 REFERENCES timings(InstrumentFamily)

```

Adding a foreign key for Families

The 2 tables above need to contain static data. I populated them using phpMyAdmin's interface, as per the TCL regulations specified in section [1.1.2](#) and [1.1.3](#). All data was entered in lowercase. Any inputs will be converted to lowercase before querying the database to make the process case-insensitive and more robust.

Server: MySQL 3306 » Database: soundbrodb » Table: timings

Browse Structure SQL Search Insert Export Import Privileges Operations Triggers

✓ 2 rows inserted.

```
INSERT INTO `timings` (`Grade`, `InstrumentFamily`, `Duration`) VALUES ('5', 'drum kit and percussion', '21'), ('6', 'drum kit and percussion', '27');
```

[ Edit inline ] [ Edit ] [ Create PHP code ]

Column	Type	Function	Null	Value
Grade	int			7
InstrumentFamily	varchar(191)			drum kit and percussion
Duration	int			27

Go

*Entering data into Timings*

	← T →	Grade	InstrumentFamily	Duration
<input type="checkbox"/>	Edit  Copy  Delete	6	guitar	23
<input type="checkbox"/>	Edit  Copy  Delete	7	guitar	23
<input type="checkbox"/>	Edit  Copy  Delete	8	guitar	28
<input type="checkbox"/>	Edit  Copy  Delete	1	strings	13
<input type="checkbox"/>	Edit  Copy  Delete	2	strings	13
<input type="checkbox"/>	Edit  Copy  Delete	3	strings	13
<input type="checkbox"/>	Edit  Copy  Delete	4	strings	18
<input type="checkbox"/>	Edit  Copy  Delete	5	strings	18
<input type="checkbox"/>	Edit  Copy  Delete	6	strings	23
<input type="checkbox"/>	Edit  Copy  Delete	7	strings	23

*Some data stored in Timings*

✓ 2 rows inserted.

```
INSERT INTO `families` (`Instrument`, `InstrumentFamily`) VALUES ('french horn', 'brass'), ('eb tenor horn', 'brass');
```

[ Edit inline ] [ Edit ] [ Create PHP code ]

Column	Type	Function	Null	Value
Instrument	varchar(191)			trumpet
InstrumentFamily	varchar(191)			brass

Go

Entering data into Families

	<input type="checkbox"/>	<input type="checkbox"/> Edit	<input type="checkbox"/> Copy	<input type="checkbox"/> Delete	Instrument	InstrumentFamily
	<input type="checkbox"/>				eb bass	brass
	<input type="checkbox"/>				bb bass	brass
	<input type="checkbox"/>				acoustic guitar	guitar
	<input type="checkbox"/>				classical guitar	guitar
	<input type="checkbox"/>				violin	strings
	<input type="checkbox"/>				viola	strings
	<input type="checkbox"/>				cello	strings
	<input type="checkbox"/>				double bass	strings
	<input type="checkbox"/>				scottish traditional fiddle	strings
	<input type="checkbox"/>				singing	singing
	<input type="checkbox"/>				flute	woodwind
	<input type="checkbox"/>				clarinet	woodwind

### *Some data stored in Families*

Using SQL, I tested the relationship between the 2 tables to return the duration of exams given the instrument and grade. Using SQL's JOIN command, the tables seemed to be working fine.

The screenshot shows a MySQL query results page. At the top, a green bar indicates "Showing rows 0 - 0 (1 total, Query took 0.0011 seconds.)". Below this is the SQL query:

```
SELECT Instrument,Grade,Duration FROM families JOIN timings ON families.InstrumentFamily=timings.InstrumentFamily WHERE Instrument = 'acoustic guitar' and grade=2;
```

Below the query are several buttons: Profiling [Edit inline] [Edit] [Explain SQL] [Create PHP code] [Refresh]. Underneath are filtering options: Show all | Number of rows: 25 | Filter rows: Search this table. A "Extra options" button is also present. The main result table has columns: Instrument, Grade, Duration. One row is shown:

Instrument	Grade	Duration
acoustic guitar	2	13

*Testing the relationship between Families and Timings.*

The screenshot shows a MySQL query results page. At the top, a green bar indicates "Showing rows 0 - 0 (1 total, Query took 0.0011 seconds.)". Below this is the SQL query:

```
SELECT Instrument,Grade,Duration FROM families JOIN timings ON families.InstrumentFamily=timings.InstrumentFamily WHERE Instrument = 'drum kit' and grade=7;
```

Below the query are several buttons: Profiling [Edit inline] [Edit] [Explain SQL] [Create PHP code] [Refresh]. Underneath are filtering options: Show all | Number of rows: 25 | Filter rows: Search this table. A "Extra options" button is also present. The main result table has columns: Instrument, Grade, Duration. One row is shown:

Instrument	Grade	Duration
drum kit	7	27

*Testing this relationship with different data.*

As these images show, the duration returned matches the TCL regulations in section 1.1.2. As per my entity relationship diagram and detailed database design outlined in section [2.4 Data](#), I continued to create the rest of the tables.

Server: MySQL 3306 » Database: soundbrodb » Table: students

	#	Name	Type	Collation	Attributes	Null	Default	Comments	Extra
□	1	StudentID	int			No	None		AUTO_INCREMENT
□	2	FirstName	varchar(191)	utf8mb4_0900_ai_ci		No	None		
□	3	LastName	varchar(191)	utf8mb4_0900_ai_ci		No	None		
□	4	Email	varchar(191)	utf8mb4_0900_ai_ci		No	None		
□	5	Password	varchar(191)	utf8mb4_0900_ai_ci		No	None		
□	6	ParentEmail	varchar(191)	utf8mb4_0900_ai_ci		No	None		

Students table

Server: MySQL 3306 » Database: soundbrodb » Table: supervisors

	#	Name	Type	Collation	Attributes	Null	Default	Comments	Extra
□	1	SupervisorID	int			No	None		AUTO_INCREMENT
□	2	FirstName	varchar(191)	utf8mb4_0900_ai_ci		No	None		
□	3	LastName	varchar(191)	utf8mb4_0900_ai_ci		No	None		
□	4	Email	varchar(191)	utf8mb4_0900_ai_ci		No	None		
□	5	Password	varchar(191)	utf8mb4_0900_ai_ci		No	None		

Supervisors table

Server: MySQL 3306 » Database: soundbrodb » Table: basicinfo

	#	Name	Type	Collation	Attributes	Null	Default	Comments	Extra
□	1	StartDate	date			No	None		
□	2	Timestamp	timestamp			No	CURRENT_TIMESTAMP		DEFAULT_GENERATED
□	3	IsPublished	tinyint(1)			No	0		

BasicInfo table

Server: MySQL 3306 » Database: soundbrodb » Table: examaccomp

	#	Name	Type	Collation	Attributes	Null	Default	Comments	Extra	Action
<input type="checkbox"/>	1	AccomplID	int			No	None			Change  Drop  More
<input type="checkbox"/>	2	School	tinyint(1)			No	None			Change  Drop  More
<input type="checkbox"/>	3	Start0900	tinyint(1)			No	None			Change  Drop  More
<input type="checkbox"/>	4	Start0930	tinyint(1)			No	None			Change  Drop  More
<input type="checkbox"/>	5	Start1000	tinyint(1)			No	None			Change  Drop  More
<input type="checkbox"/>	6	Start1030	tinyint(1)			No	None			Change  Drop  More
<input type="checkbox"/>	7	Start1100	tinyint(1)			No	None			Change  Drop  More
<input type="checkbox"/>	8	Start1130	tinyint(1)			No	None			Change  Drop  More
<input type="checkbox"/>	9	Start1200	tinyint(1)			No	None			Change  Drop  More
<input type="checkbox"/>	10	Start1230	tinyint(1)			No	None			Change  Drop  More
<input type="checkbox"/>	11	Start1300	tinyint(1)			No	None			Change  Drop  More
<input type="checkbox"/>	12	Start1330	tinyint(1)			No	None			Change  Drop  More
<input type="checkbox"/>	13	Start1400	tinyint(1)			No	None			Change  Drop  More
<input type="checkbox"/>	14	Start1430	tinyint(1)			No	None			Change  Drop  More
<input type="checkbox"/>	15	Start1500	tinyint(1)			No	None			Change  Drop  More
<input type="checkbox"/>	16	Start1530	tinyint(1)			No	None			Change  Drop  More
<input type="checkbox"/>	17	Start1600	tinyint(1)			No	None			Change  Drop  More
<input type="checkbox"/>	18	Start1630	tinyint(1)			No	None			Change  Drop  More

ExamAccomps table

```

1 ALTER TABLE examinees
2 ADD CONSTRAINT examinees_fk_1
3 FOREIGN KEY (ExamineeID)
4 REFERENCES students(StudentID),
5
6 ADD CONSTRAINT examinees_fk_2
7 FOREIGN KEY (Instrument)
8 REFERENCES families(Instrument),
9
10 ADD CONSTRAINT examinees_fk_3
11 FOREIGN KEY (Grade)
12 REFERENCES timings(Grade),
13
14 ADD CONSTRAINT examinees_fk_4
15 FOREIGN KEY (SupervisorID)
16 REFERENCES supervisors(SupervisorID),
17
18 ADD CONSTRAINT examinees_fk_5
19 FOREIGN KEY (AccompID)
20 REFERENCES examaccompns(AccompID);
21

```

*Adding foreign keys to examinees using SQL*

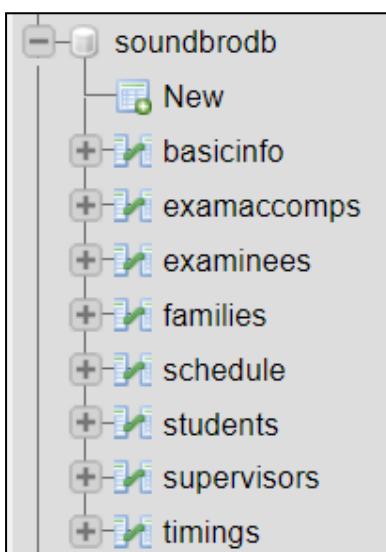
Table: examinees									
	#	Name	Type	Collation	Attributes	Null	Default	Comments	Extra
<input type="checkbox"/>	1	ExamineeID 	int			No	None		
<input type="checkbox"/>	2	Instrument 	varchar(191)	utf8mb4_0900_ai_ci		No	None		
<input type="checkbox"/>	3	Grade 	int			No	None		
<input type="checkbox"/>	4	SupervisorID 	int			No	None		
<input type="checkbox"/>	5	AccompID 	int			Yes	NULL		

*Examinees table*

The screenshot shows the 'soundbrodb' database in phpMyAdmin. The 'schedule' table is selected. The table structure is as follows:

#	Name	Type	Collation	Attributes	Null	Default	Comments	Extra
1	ScheduleID	int			No	None		AUTO_INCREMENT
2	ExamineeID	int			No	None		
3	StartTime	time			No	None		

*Schedule table*



All of the tables have now been created, giving the structure shown in the image to the left. Although it is not clear in the images above, I also used SQL to manually create each foreign key. I created a unique index for Students.Email and Supervisors.Email . phpMyAdmin does not store a boolean field as 'boolean', but instead calls the data type 'tinyint(1)'. This means an integer between 0 and 1 (ie. 0 or 1). This allows me to use the field in the same way as I would a boolean field. For the relevant fields, I created auto-incrementing primary keys. For other fields, such as BasicInfo.IsPublished, I defined the default value. This default value is inserted into the database if no value is given. This is useful in BasicInfo, as the field Timestamp should store the current timestamp, and IsPublished should store False (ie. 0).

*Database structure*

The database should now be complete. Whenever the tables need to be cleared, I will use TRUNCATE instead of DROP. To test the functionality of the database, I created some fake data to input into the database. Although I could input the real data provided by Ms Pearcey, I would rather use this later in my development, such as for system tests. Using an online name generator, I created some names for the Students and Supervisors tables. I then used this data to then populate the rest of the database. The data follows the same patterns, for example every user's email is `firstname.lastname@email.com` . This is for simplicity purposes, as I am just testing how the relational database processes some requests. All data is randomly generated, and any similarities are purely coincidental. Once all the data was inserted into the database, I carried out the tests below. I have not generated fake data for the Schedule table, as it did not seem necessary to test this table until later. Please note that, in this fake example, the password is written in plain text. With the real system, the password hash is stored in the database.

I decided to test the database by making queries that are likely to be used later. The first test is to retrieve the duration of an examinee, given their examineeID. This will test how successfully Examinees, Families and Timings connect with each other. I selected `examineeID = 3` for these tests.

		ExamineeID	Instrument	Grade	SupervisorID	AccompID
<input type="checkbox"/>	Edit  Copy  Delete	1	piano	4	8	3
<input type="checkbox"/>	Edit  Copy  Delete	2	violin	2	10	NULL
<input type="checkbox"/>	Edit  Copy  Delete	3	classical guitar	6	4	1

Examinees table with fake data

This portion of the examinees table shows that ExamineeID=3 is playing the classical guitar at grade 6. According to regulations, the duration for this exam is 23 minutes (see figure 1.1.2b). Therefore, the expected output is 23 minutes.

Showing rows 0 - 0 (1 total, Query took 0.0119 seconds.)						
<pre>SELECT ExamineeID,examinees.Grade,examinees.Instrument,families.InstrumentFamily,Duration FROM timings JOIN families ON families.InstrumentFamily = timings.InstrumentFamily JOIN examinees ON examinees.Instrument = families.Instrument AND timings.Grade=examinees.Grade WHERE examineeID=3;</pre>						
<input type="checkbox"/> Profiling [ Edit inline ] [ Edit ] [ Explain SQL ] [ Create PHP code ] [ Refresh ]						
<input type="checkbox"/> Show all	Number of rows:	25	Filter rows:	Search this table		
<input type="checkbox"/> Extra options						
ExamineeID	Grade	Instrument	InstrumentFamily	Duration		
3	6	classical guitar	guitar	23		

SQL result of first test

This image shows the SQL used to make the query to the database, and its result. The duration of 23 minutes is correct, therefore it has passed this test. Another common query is to find the name of an examinee's accompanist. The user with examineeID=3 has AccomplID=1, which is equivalent to SupervisorID=1. This will test how Examinees, Supervisors and ExamAccomps can be joined together to give correct results.

SupervisorID	FirstName	LastName	Email	Password
1	David	Williams	david.williams@email.com	Password_williams1
2	Sarah	Johnson	sarah.johnson@email.com	Password_johnson1
3	Christopher	Smith	christopher.smith@email.com	Password_smith1
4	Amy	Jones	amy.jones@email.com	Password_jones1

ExamAccomps data

When retrieving the name of the accompanist for examineeID=3, the output should be David Williams.

 Showing rows 0 - 0 (1 total, Query took 0.0047 seconds.)					
<pre>SELECT FirstName,LastName FROM supervisors JOIN examaccompson ON supervisors.SupervisorID = examaccompson.AccompID JOIN examinees ON examaccompson.AccompID = examinees.AccompID WHERE examineeID = 3;</pre>					
<input type="checkbox"/> Profiling [ Edit inline ] [ Edit ] [ Explain SQL ] [ Create PHP code ] [ Refresh ]					
<input type="checkbox"/> Show all   Number of rows: 25 <input type="button" value="▼"/> Filter rows: <input type="text" value="Search this table"/>					
<input type="button" value="Extra options"/>					
<table border="1"> <thead> <tr> <th>FirstName</th><th>LastName</th></tr> </thead> <tbody> <tr> <td>David</td><td>Williams</td></tr> </tbody> </table>		FirstName	LastName	David	Williams
FirstName	LastName				
David	Williams				

### SQL result of second test

This image clearly shows that ‘David Williams’ has been outputted using the SQL at the top of the image. This test was also successful. I will conduct one more test to assess the functionality of this program. The above images show that the teacher for ExamineeID=3 has a SupervisorID of 4, which is Amy Jones, or amy.jones@email.com. The screenshot below shows that ExamineeID=3 is Alex Johnson, or alex.johnson@email.com. Given the students email, I will run an SQL query to find the teachers email. This will test the relationships between the tables: Students, Examinees and Supervisors. The expected output is [amy.jones@email.com](mailto:amy.jones@email.com).

StudentID	FirstName	LastName	Email	Password	ParentEmail
3	Alex	Johnson	alex.johnson@email.com	Password_johnson1	johnson.parent@email.com

### Students data

 Showing rows 0 - 0 (1 total, Query took 0.0025 seconds.)			
<pre>SELECT supervisors.Email FROM supervisors JOIN examinees ON supervisors.SupervisorID = examinees.SupervisorID JOIN students ON examinees.ExamineeID = students.StudentID WHERE students.Email="alex.johnson@email.com";</pre>			
<input type="checkbox"/> Profiling [ Edit inline ] [ Edit ] [ Explain SQL ] [ Create PHP code ] [ Refresh ]			
<input type="checkbox"/> Show all   Number of rows: 25 <input type="button" value="▼"/> Filter rows: <input type="text" value="Search this table"/>			
<input type="button" value="Extra options"/>			
<table border="1"> <thead> <tr> <th>Email</th></tr> </thead> <tbody> <tr> <td>amy.jones@email.com</td></tr> </tbody> </table>		Email	amy.jones@email.com
Email			
amy.jones@email.com			

### SQL result of third test.

Given the appropriate SQL query, ‘amy.jones@email.com’ was returned, matching the expected output. Therefore, this test was successful. As a result of these tests, I am confident that the database has been designed and implemented correctly. As it has been normalised to third normal form, SQL queries give reliable results.

### 3.1.2 Review of stage 1

#### *Stage 1 summary*

This stage was relatively small, and involved no programming, except for conducting some tests using SQL. However, it was a crucial step in the project. In this stage:

- All tables in the database have been created, and have been tested to ensure that the database structure and implementation is successful.
- I populated the database with fake data for testing purposes.

#### *Testing carried out*

- Returned the duration of an examinee's exam, given an examineeID
- Returned the name of an examinee's accompanist, given an examineeID
- Returned the email of an examinee's teacher, given the student's email

#### *Success criteria met*

None of the success criteria have been met explicitly, but this section has enabled many success criteria to be met in later stages. For example, I3 and S8 specify data being stored in the server, so can only be met if this stage is successful. In addition, almost all success criteria would not be met without this stage. This includes L1-10, E1-3, and I1-8, as the key functionality of this solution depends on a well-structured database.

#### *Changes to the design*

One minor change has been made: tables will not be deleted at the end and created at the start of each cycle. Instead, the TRUNCATE command in SQL will be used to clear all data from the table without deleting the table itself. This is better for performance, gives the same results, and requires simpler code.

## 3.2 Login functionality

### 3.2.1 Developing

This stage will involve developing the login feature of this project. This will allow students and supervisors to log in to the system and access their timetable on the dashboard. However, this stage will only focus on the process of logging them in, not displaying the schedule. This will also allow Ms Pearcey to schedule exams securely, as her email will be recognised upon login, providing access to all scheduling features. Only her email will allow access to these features. When the solution is implemented in real life, her actual email will be hard-coded into the system. Until then, admin@email.com will represent Ms Pearcey's login, and will be the only login to allow access to scheduling features.

In section 3.1 I created fake data to populate the database. I added to this database to create 3 important records for testing purposes. For these records, the Password field stores the hash of the password using PHP's `password_hash()` function.

SupervisorID	1	FirstName	LastName	Email	Password
17	John	Doe		john.doe@email.com	\$2y\$10\$EDsmwRoI23DU/PfHaTKs4.pcAAchICVtjCbyiQ9R142...
16	Ad	Min		admin@email.com	\$2y\$10\$QOlo/wWG0fTaVByyWmZz.OZQO6cPQO9ZchjUOJdJHTI...

Test records for Supervisors table

StudentID	1	FirstName	LastName	Email	Password	ParentEmail
31	Stu	Dent		student@email.com	\$2y\$10\$vqZXxY2XDSdhQftfwH5Bu12Vo.HEN51N3YWAekOihx...	parent@email.com

Test records for Student table

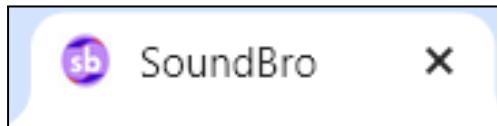
The correct password for John Doe is Johndoe1. The correct password for Ad Min is Admin123. The correct password for Stu Dent is Student123. These will be used throughout this stage for iterative testing. Firstly, I developed the index page, `index.php`. This was styled using Bootstrap 5 classes.

```
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1, shrink-to-fit=no">
    <link href='https://fonts.googleapis.com/css?family=League Spartan' rel='stylesheet'>
    <link rel="stylesheet" href="https://cdn.jsdelivr.net/npm/bootstrap-icons@1.11.2/font/bootstrap-icons.min.css">
    <link href="https://cdn.jsdelivr.net/npm/bootstrap@5.3.2/dist/css/bootstrap.min.css" rel="stylesheet" integrity="sha384-T3c6CoIi6uLra9TneNEoa7RxnatzjcDSCmG1MXxSR1GAsXEV/Dwykc2MPK8M2HN" crossorigin="anonymous">
    <link href='styles.css' rel='stylesheet'>
    <link rel="icon" href="logo/favicon3.png">
    <title>SoundBro</title>
</head>
```

`index.php <head> tag`

The `<meta>` tags allow the website to adjust to the screen size of the device, making it more compatible with mobile devices. The first `<link>` tag enables the SoundBro logo to use the League Spartan font, ensuring consistency. The second `<link>` tag connects to the Bootstrap icons, the third connects to the CSS for styling with Bootstrap. Finally, the page links to

styles.css. For usability, I have included a favicon and title, and plan to include this on each page, with the title representing the purpose of each page.



*favicon and title for index.php*

styles.css is an additional stylesheet, which I have developed myself. This allows me to adjust the styling of the web page exactly as I would like, to ensure it contains the usability features from section [2.3](#).

```
/* format image to be displayed correctly */
.biglogo{
    display:block;
    max-width:60%;
    margin: 20px auto;
    padding:20px;
}

/*Same font as SoundBro logo for consistency*/
.logo{
    font-family:'League Spartan';
    letter-spacing: -1px;
}
```

*Styling specific features in styles.css*

Currently, this file contains the appropriate styling so that index.php is displayed correctly. The class .logo ensures that the SoundBro logo is displayed as it has been designed, ensuring the logo is consistent and allows the product to become more familiar with users. This <head> tag will be used on every page to ensure consistency, but the title will be changed each time.

Next, I developed the navigation bar. This should only display the appropriate links, so if logged out then 'dashboard', 'login' and 'register' should be displayed. If logged in, 'dashboard' and 'logout' will be displayed. Throughout my system, the session variable \$\_SESSION['user\_id'] will store the StudentID/SupervisorID of the current user. Depending on if the variable has been set or not, the appropriate links will be displayed in the navigation bar.

```

<nav class="navbar navbar-expand-sm bg-primary navbar-dark">
    <!-- display links for navbar -->
    <ul class="navbar-nav">
        <li class="nav-item">
            <a class="nav-link logo fw-bolder fs-4 navbar-brand ms-2" href="index.php">SoundBro.</a>
        </li>
        <li class="nav-item">
            <a class="nav-link me-2" href="dashboard.php"><i class="bi bi-music-note-list"></i> Dashboard</a>
        </li>
        <?php
        // Check if the user is logged in
        if (isset($_SESSION['user_id'])) {
            echo '<li class="nav-item"><a href="logout.php" class="nav-link me-2"><i class="bi bi-box-arrow-right"></i> Logout</a></li>';
        } else {
            echo '<li class="nav-item"><a class="nav-link me-2" href="login.php"><i class="bi bi-box-arrow-in-right"></i> Login</a></li>';
            <li class="nav-item"><a class="nav-link me-2" href="register.php"><i class="bi bi-pen"></i> Register</a></li>';
        }
        ?>
    </ul>
</nav>

```

### *Navigation bar code*

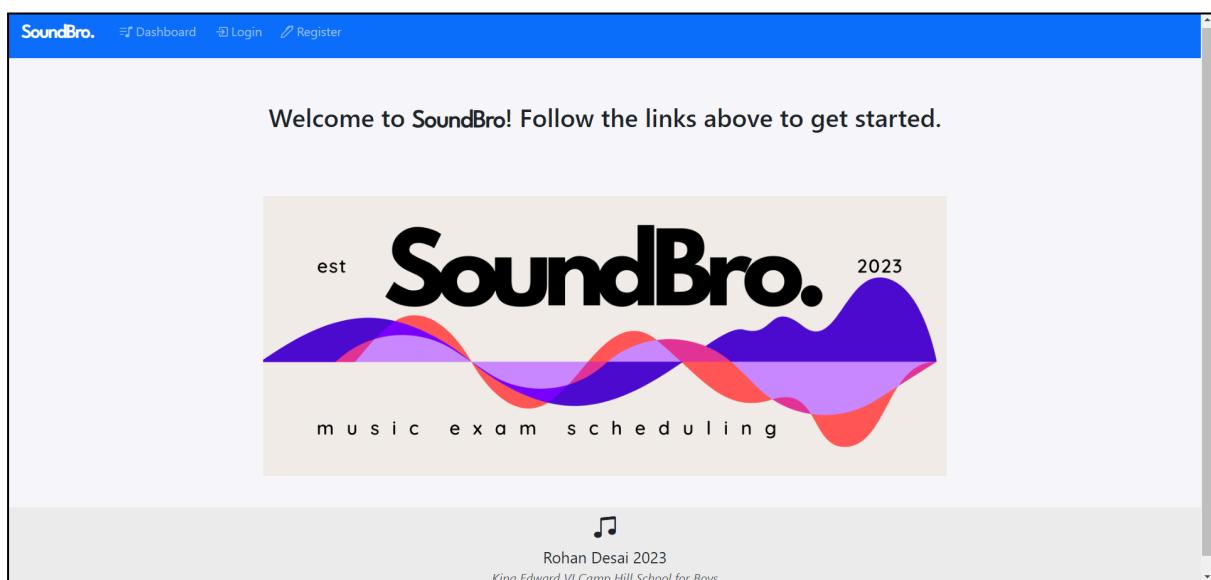
This code was originally stored in index.php, however I realised that the same navigation bar must be used across every webpage. This ensures consistency across each page, making each page feel familiar to the user and meeting the usability features from section [2.3](#). To achieve this, I stored the above code in a separate file called navbar.php, and referenced this in my main file. I also did the same for my footer, which once again ensures consistency.

```
<?php include 'includes/navbar.php' ?>
```

*Referencing the navbar (this will be on each page)*

```
<?php include 'includes/footer.html'?>
```

*Referencing the footer (this will be on each page)*



index.php displays this page, which I believe meets the usability requirements. The main logo has been displayed here, which gives the user a good introduction to the system. The SoundBro text in the navigation bar will link to this page. To test the functionality of the navigation bar, I set `$_SESSION['user_id']=1`.



This change resulted in the navigation bar successfully changing, so this feature is successful.

Next, I developed the login page. Using HTML and Bootstrap CSS I developed the functionality to submit data to the server. The PHP code then processes this request to determine if the user should be logged in or not.

```
// connect to database
$serverName = "localhost";
$serverUsername = "root";
$serverPassword = "";
$dbname = "soundBroDB";
$conn = new mysqli($serverName, $serverUsername, $serverPassword, $dbname);

// unexpected fatal error
if ($conn->connect_error) {
| die("Connection failed: " . $conn->connect_error);
}
```

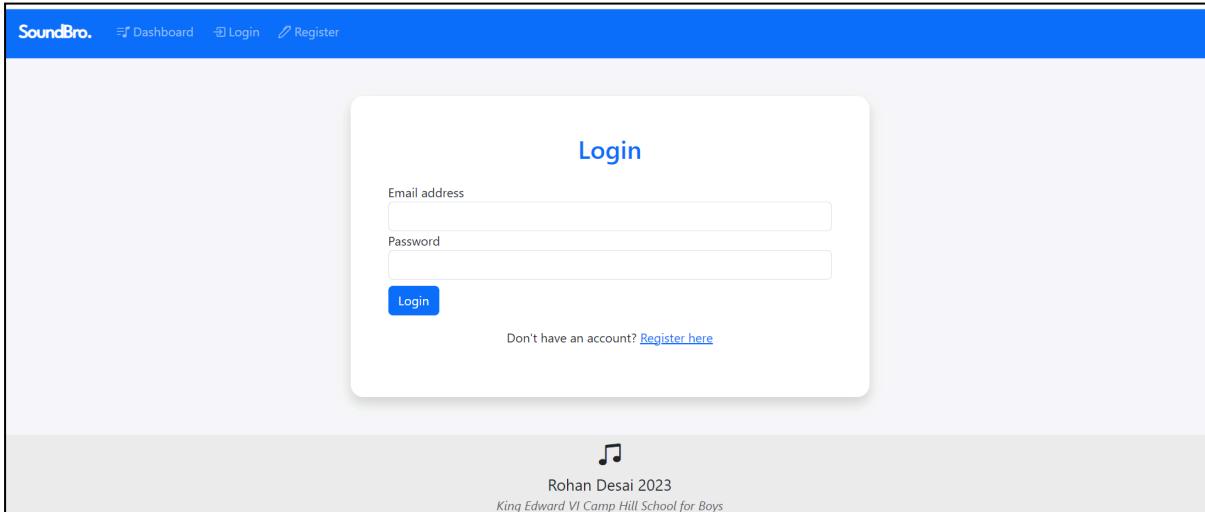
Almost every page will interact with the database. This is done using the MySQLi class with the appropriate credentials to connect to the right database. This class allows a connection (`$conn`) to the database to be set up. As this will be used on every page, I have again used PHP ‘include’ to reference the appropriate file for this data.

```
// connect to database
include 'includes/db-connect.php';
```

As many immature students may try to gain unauthorised access to certain pages, it is imperative that each page verifies who is accessing the page, and redirects anyone who should not have access to that page. This method also helps users by redirecting them to the correct page. In this case, if someone is already logged in, they should be redirected to the dashboard.

```
// If logged in, redirect to the dashboard
if (isset($_SESSION['user_id'])) {
    header("Location: dashboard.php"); // Redirect to dashboard
    exit();
}
```

Next, I created the main element of the login page. This involved using <form> to allow users to enter and submit details. To prevent any confusion, I included a helpful message which links to the registration page (yet to be developed).



As this page references the same navigation bar and footer, it is consistent with index.php. Once data is submitted, all inputs must be sanitised to prevent SQL injection. Then, many validation checks occur to provide specific, helpful error messages. This is completed through a series of if else statements. I have followed the pseudocode designed for the login process in section [2.5.3](#).

```
// validation - presence check
if ($email=="" or $password==""){
    $error="You must enter a username and password";
} else {
```

The first check is a presence check, so an error is raised if nothing is submitted. If this does not raise an error, the email is validated against RFC 822 using the function filter\_var, as described in [2.4](#). If no error is found, the email is valid. The next step is to verify the password. I had some difficulty in searching through 2 tables (Students and Supervisors) to check the password. Firstly, I have searched through both tables and stored the results of both queries separately. This allows me to identify any unexpected errors when querying the database.

```

// Fetch login info for both students and supervisors
$studentQuery = "SELECT StudentID,Password FROM Students WHERE Email = '$email'";
$supervisorQuery = "SELECT SupervisorID,Password FROM Supervisors WHERE Email = '$email'";
$studentResult = $conn->query($studentQuery);
$supervisorResult = $conn->query($supervisorQuery);

if (!$studentResult or !$supervisorResult) {
    $error = "Could not retrieve database info: " . $conn->error . ". Try again later.";
} else {

```

Once both queries have been completed, the result from the Students table (\$studentResult) is checked. The order in which the tables are checked makes no difference, so I could have checked Supervisors first instead.

The function password\_hash will hash the password so that it can be appropriately compared to the stored hash of the password in the database. If the passwords are equal, the retrieved StudentID is set as a session variable. Also, the user type is defined. This will be helpful for future web pages such as the dashboard. It can also be useful for identifying if a user should have access to a particular page. If login is successful, they are redirected to the dashboard. If not, \$error stores “invalid password”.

If the email is not found in Students, it checks through Supervisors using the same process. However, if it is not found here, an error message “Email not found” should be displayed. If the email has been found and the password is correct, it will redirect the user to the dashboard. However, there is one important check to determine the user type. As Ms Pearcey’s details will be stored in Supervisors, the code below will check if she is logging in. If so, the user type is ‘admin’. This gives her exclusive access to all scheduling functionality.

```

if ($email=='admin@email.com') {
    // admin login (Ms Pearcey)
    $_SESSION['user_type']='admin';
} else {
    $_SESSION['user_type']='supervisor';
}

```

At any point, if an error is found, the message is stored in the variable \$error. In the frontend of login.php, I have included the code below. This will check if \$error has been set. If it has, it will display the error clearly in a red box. I decided to use this method rather than using die() or some other function, as it gives me much more control over how the error will appear on the page. It does not shock or overwhelm the user, and does not appear in the corner of the page. Using Bootstrap classes, the box can be styled to look professional and alert the user effectively.

```

<!-- display error -->
<?php if (isset($error)) { ?>
    <div class="alert alert-danger">
        <?php echo $error; ?>
    </div>
<?php } ?>

```

Once logged in, the user will be redirected to the dashboard, dashboard.php. I have developed this page, which will simply display the user's first name in a welcome message. The dashboard will eventually display the relevant part of the schedule for all users. However, if Ms Pearcey logs in, she will be displayed another message. As she is a supervisor and an admin, the dashboard will display supervisor-related information. However, she will also see a link to the admin dashboard, where she can schedule exams.

```

<h1 class ="p-2 mt-5 text-center">Dashboard</h1>
<h3 class ="mb-5 text-center text-secondary">Welcome, <?php echo $firstName ; ?>! </h3>
<!-- display additional message for Ms Pearcey (admin) -->
<?php if ($_SESSION['user_type']=='admin') { ?>
    <div class="container p-3 my-5 border">
        <p class="fs-5">
            Welcome! If you are here for admin purposes, click <a href="dashboard-admin.php">here.</a> <br>
            If not, stay on this page to see music teacher / accompanist info.
        </p>
    </div>
<?php } ?>

```

The session variable \$\_SESSION['user\_type'] will allow the program to distinguish between Ms Pearcey and other users. Another session variable, \$\_SESSION['user\_id'], will be used to distinguish what data should be displayed to who. This will store the StudentID/SupervisorID of the current user. If no variable has been set, the page redirects back to the login page.

I have also developed a page called logout.php . This will be a link in the navigation bar, displayed to users that have been logged in. It deletes all session variables, ends the session, and redirects back to index.php . This is shown below.

```

session_start();
session_unset(); // unset all session variables
session_destroy(); // destroy the session
header("Location: index.php"); // Redirect to index page
exit();

```

### 3.2.2 Unit testing

I entered a wide range of data to appropriately test the full range of this feature. This involved valid and invalid data (cannot enter boundary data for this), and data to return every specific output in order to test every part of this function appropriately. For each test, I have created a row of a test table, and below it I have included a screenshot of the result.

Purpose of test	Test data	Expected Result	Successful?
Testing the presence check (invalid data)	Null (nothing entered)	"You must enter a username and password"	Yes

The screenshot shows a login form with the title "Login". It has two input fields: "Email address" and "Password", both of which are empty. Below the inputs is a blue "Login" button. A red error message box at the top contains the text "You must enter a username and password". At the bottom of the page, there is a link "Don't have an account? [Register here](#)".

Purpose of test	Test data	Expected Result	Successful?
Testing the email validation with invalid data	Email = "a@b" Password = "test"	"Invalid email format"	Yes

The screenshot shows a login form with the title "Login". It has two input fields: "Email address" and "Password", both of which are empty. Below the inputs is a blue "Login" button. A red error message box at the top contains the text "Invalid email format". At the bottom of the page, there is a link "Don't have an account? [Register here](#)".

When using `<input type="email">`, validation is automatically carried out to detect some invalid email addresses. However, using `filter_var()` is more rigorous as it validates against RFC 822, so will catch almost all invalid email addresses. The above test was not deemed invalid by the input tag, but was invalid according to `filter_var()`. The test below shows an example of the input tag's validation catching invalid emails.

Purpose of test	Test data	Expected Result	Successful?
Testing the HTML email validation with invalid data	email= "abc" password= "test"	Please include an "@" in the email address. 'abc' is missing an '@'.	Yes

The screenshot shows a 'Login' page with two input fields: 'Email address' containing 'abc' and 'Password' containing '....'. A validation message 'Please include an '@' in the email address. 'abc' is missing an '@.' is displayed next to the password field. A 'Login' button and a 'Register here' link are also visible.

Purpose of test	Test data	Expected Result	Successful?
Testing with an invalid student password	email= "student@email.com" password= "test"	"Incorrect password"	Yes

The screenshot shows a 'Login' page with an 'Email address' field and a 'Password' field, both currently empty. A red error message 'Incorrect password' is displayed above the fields. A 'Login' button and a 'Register here' link are also visible.

Purpose of test	Test data	Expected Result	Successful?
-----------------	-----------	-----------------	-------------

Testing with a valid student password (see start of 3.2 for valid logins)	email= "student@email.com" password= "Student123"	Redirected to dashboard.php	No
---	--	-----------------------------	----

The screenshot shows a login form with a pink error message box at the top containing the text 'Incorrect password'. Below the message box are two input fields: 'Email address' and 'Password'. At the bottom left is a blue 'Login' button, and at the bottom right is a link 'Don't have an account? [Register here](#)'.

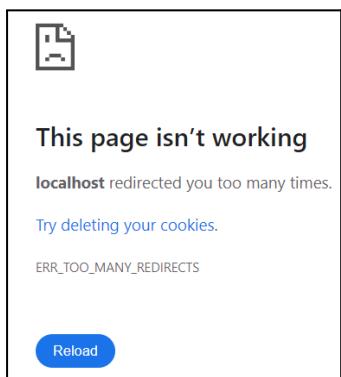
This test was unsuccessful, so I looked closer at my code to examine why. This login has correctly passed the presence check, and email validation check. The email has been found in the Students table (otherwise the message would say 'email not found'). This record only exists in the Students table, not in the Supervisors table. Additionally, the message clearly shows that the password has been found to be incorrect. Therefore, the error must be involved in comparing the hashed password to the entry in the database. I confirmed that the data was entered into the database using the same function as the one used here, `password_hash($password,PASSWORD_DEFAULT)`. I could not deduce what the issue was, as I had the basic understanding that hashing is a deterministic process, so the same input used with the same hashing algorithm must give the same result. However, I used the PHP echo function to print the variables to the screen, where I realised that the same input outputted a different hash every single time. I researched further into the `password_hash()` function and learned that it also uses a process called 'salting'. This adds a random value to the input before hashing, known as the salt. This causes the same input to give a different output, making the process more complex and secure. This is when I found that the function `password_verify()` can check if a password matches its hash, even if salting is used. I adjusted my code so that the inputted password was no longer hashed, and the if statement used the `password_verify()` function.

```
$passwordDB=$row[ 'Password '];

// Verify the password
if (password_verify($password,$passwordDB)) {
|
```

I made this change for checking the password in the Students table **and** in the Supervisors table. Then, I ran the same test again.

Purpose of test	Test data	Expected Result	Successful?
Testing with a valid student password	email= "student@email.com" password= "Student123"	Redirected to dashboard.php	No

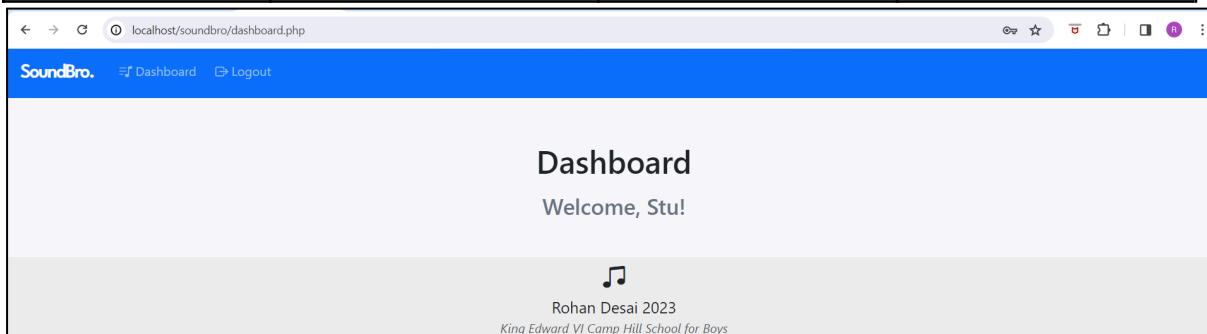


This error shows that the password was accepted, as it redirected me to dashboard.php. Therefore, the previous change was successful. However, this means that the dashboard redirected me back to the login page, and the 2 pages redirected back and forth until an error was returned. If the dashboard tried to redirect me back, it must not have received the session variable `$_SESSION['user_id']`. After some research, I understood that the line `session_start()` must be added to the top of every page in order to access these variables.

```
session_start();
```

I added this to the top of dashboard.php, and ran the same test again.

Purpose of test	Test data	Expected Result	Successful?
Testing with a valid student password	email= "student@email.com" password= "Student123"	Redirected to dashboard.php	Yes

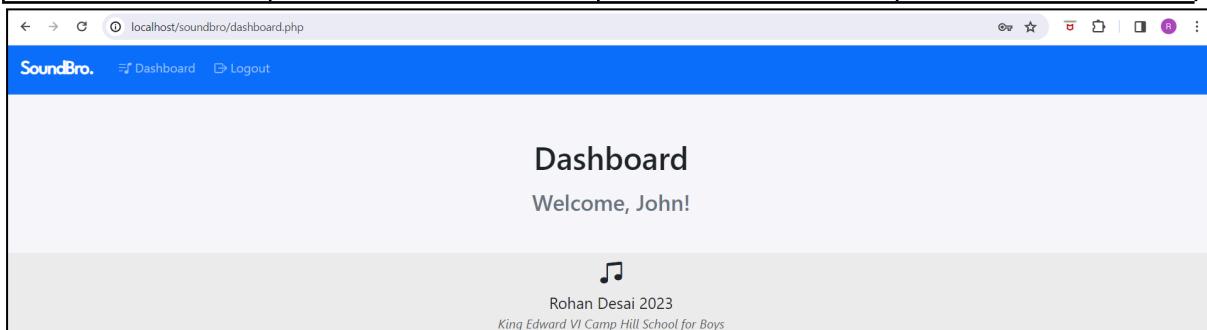


This shows that the session variables have been transferred to this page, as it has not redirected. Also, the user\_id and the user\_type has been used to retrieve the user's first name from the database, showing that this process is working fine.

Purpose of test	Test data	Expected Result	Successful?
Check the search for an email across both tables (invalid data)	email= "fake@email.com" password= "test"	"Email not found"	Yes

The screenshot shows a login form with a red error message box at the top containing the text "Email not found". Below the message box are two input fields: "Email address" and "Password", each with a placeholder text "Enter email" and "Enter password" respectively. A blue "Login" button is located below the password field. At the bottom of the form, there is a link "Don't have an account? [Register here](#)".

Purpose of test	Test data	Expected Result	Successful?
Testing with a valid supervisor login	email = "john.doe@email.com" password = "Johndoe1"	Redirected to dashboard.php	Yes



Note that the navigation bar now displays 'dashboard' and 'logout', as the user is logged in.

Purpose of test	Test data	Expected Result	Successful?
Testing with incorrect supervisor password	email= "john.doe@email.com" password= "password123"	"Incorrect password"	Yes

The image shows a login form with a light gray background. At the top center, the word "Login" is written in blue. Below it is a red rectangular box containing the white text "Incorrect password". Underneath this box are two input fields: one for "Email address" and one for "Password", both represented by simple horizontal lines. At the bottom left is a blue "Login" button with white text. At the bottom center, there is a link in blue text that says "Don't have an account? [Register here](#)".

Purpose of test	Test data	Expected Result	Successful?
Testing with an incorrect admin password	email= "admin@email.com" password= "password123"	"Incorrect password"	Yes

The image shows a login form with a light gray background. At the top center, the word "Login" is written in blue. Below it is a red rectangular box containing the white text "Incorrect password". Underneath this box are two input fields: one for "Email address" and one for "Password", both represented by simple horizontal lines. At the bottom left is a blue "Login" button with white text. At the bottom center, there is a link in blue text that says "Don't have an account? [Register here](#)".

Purpose of test	Test data	Expected Result	Successful?

Testing with a valid admin login	email= "admin@email.com" password= "Admin123"	Redirected to dashboard.php with additional message displayed, linking to the admin dashboard.	Yes
----------------------------------	--	--	-----

The screenshot shows a 'Dashboard' page with a header 'Welcome, Ad!' and a message: 'Welcome! If you are here for admin purposes, click [here](#). If not, stay on this page to see music teacher / accompanist info.'

Please note: this is a cropped image of the dashboard, and has been cropped to clearly show the message displayed. The full dashboard looks the same as previous images. The link will take the user to 'dashboard-admin.php', which is yet to be developed.

The full code for login.php in the appendix shows that the above test goes through every single validation check / if statement. Therefore, this is valid data to show that all of the validation checks work as expected with valid data.

Purpose of test	Test data	Expected Result	Successful?
Testing logout function	Null	Redirects to index.php	Yes

The screenshot shows the SoundBro index page with a blue header bar containing the SoundBro logo, 'Dashboard', 'Login', and 'Register'. The main content area displays a welcome message: 'Welcome to SoundBro! Follow the links above to get started.' Below this is a large SoundBro logo with the text 'est 2023' and 'music exam scheduling'. At the bottom, there is a copyright notice: 'Rohan Desai 2023' and 'Kings Edward VI Camp Hill School for Boys'.

Note that the navigation bar now displays ‘dashboard’, ‘login’, and ‘register’, as the user has been logged out.

### 3.2.3 Review of stage 2

#### *Stage 2 summary*

- Developed login page, login.php
- Developed login functionality
- Developed index page, index.php
- Created dashboard, dashboard.php (basic page, all functionality on this dashboard has not been developed yet)
- Developed logout function , logout.php
- Created CSS stylesheet for specific styling, styles.css
- Also, created separate files for commonly used code: navbar.php, db-connect.php, footer.html

#### *Testing carried out*

- Tested all validation of the login feature, including verification of the password using the database
- Tested that login page redirects to dashboard
- Tested that the dashboard contains link to admin dashboard for the admin only
- Tested logout function

All tests were successful in the end.

#### *Success criteria met*

By achieving some of the usability features, I am moving closer to meeting G1.

I have achieved L1, as the system has a working login page, allowing users to enter their email and password to access their dashboard. As the login page also identifies and stores which type of user it is, and their StudentID / SupervisorID, a user will only be able to access the web pages that their credentials will allow. This means that users can only access data they are allowed to access, as long as each page will check the credentials of the user. Therefore, L2 has been met, providing that each page will include these checks.

A successful login page allows many more success criteria to be met. This includes any features displayed on the dashboard, such as L7-10. One key aspect of the ‘publish’ function is displaying specific user information, which is only possible with a login system.

#### *Changes to the design*

No major changes to the design in this section. There is a slight change that I am using the password\_verify function, instead of hashing the inputted password and comparing this to the password in the database. This does not affect anything in the project, so is a negligible change.

## *Review of the project so far*

The project now has an interface and basic structure for features to be implemented in the future. The login system allows users to access data that is only relevant to them, this feature will later be used to display specific time slots to everyone's dashboard. The success of the login feature also shows that the Students and Supervisors tables are working well. The next step is to allow users to register themselves, so that they can log on to the system without entering data manually to the database.

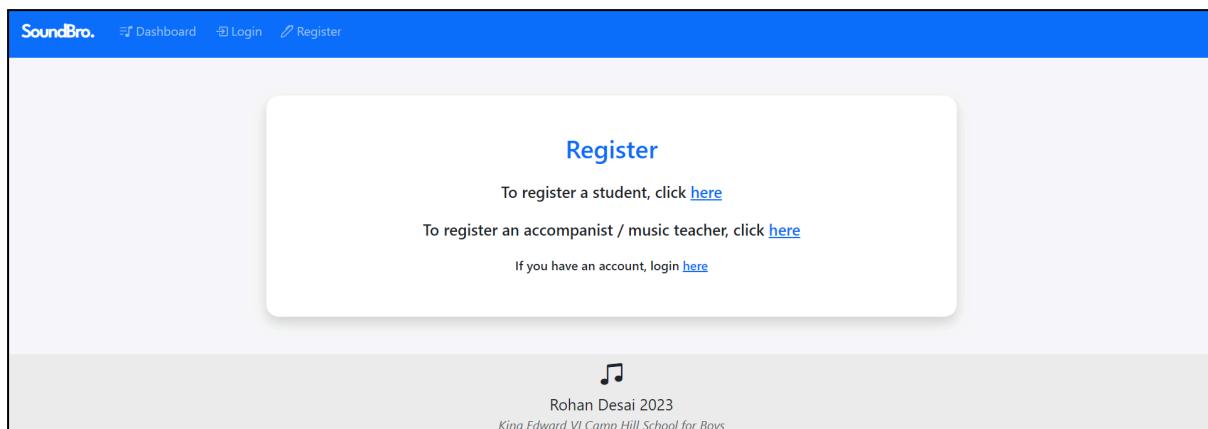
Unfortunately, my design has not specified a detailed file structure of all files in the directory. I am aware that this may be confusing for the reader. Therefore, at the end of each stage I will include a current structure of the 'soundbro' directory for clarification.

```
soundbro/
|--- index.php
|--- login.php
|--- dashboard.php
|--- logout.php
|--- styles.css
|--- includes/
    |--- db-connect.php
    |--- navbar.php
    |--- footer.html
|--- logo/
    |--- full.png (main logo on index.php)
    |--- favicon3.png (favicon)
```

## 3.3 Registration feature

### 3.3.1 Developing

To develop the registration system, I must allow both students and supervisors to log in. This is because their information will be stored in different tables. Also, they require slightly different data to be entered during registration. The Supervisors and Students tables are identical, except that Students contains an additional field of a parent's email address. As different data must be entered to be stored in different locations, I will create separate pages for registering students and supervisors. The first page, register.php, will direct users to the right direction depending on whether they are registering a student or a supervisor. This will include links to 2 pages: register-student.php and register-supervisor.php. As the pages are now separate, the program is able to understand where the data should be stored, and what data should be entered.



Using a consistent web design, this page follows a similar structure to all pages, especially the login page. This allows users to feel familiar with the software. To provide some help, I have included a smaller message directing users with an account to the login page. This makes the system easier to navigate, and overall helps my usability features to be achieved. First, I developed register-student.php.

## Register a Student

First name  
  
Names must be 2-50 characters, with no numbers or special characters (except hyphens).

Last name

Email address

Parent email address

Password  
  
Password must be at least 8 characters long and include at least one uppercase letter, one lowercase letter and one number.

Confirm Password

**Register**

If you have an account, login [here](#)

To help users with entering valid information, the names and password entry has a small message, detailing the requirements. This encourages users to enter valid data, catching any errors before they are submitted. Therefore, users are less likely to enter invalid data, so will feel more comfortable when interacting with the software, as the process will seem simple and easy.

```
<!-- enter first name -->
<div class="form-group">
  <label for="firstName">First name</label>
  <input type="text" class="form-control" id="firstName" name="firstName">
  <small class="text-muted">
    | Names must be 2-50 characters, with no numbers or special characters (except hyphens).
  </small>
</div>
```

These messages provide on-screen help, another example of a usability feature mentioned in section [2.3](#). The next step is to validate the data, using the validation specified in [2.4](#). Whenever an error is found, a specific error message is generated. This is done using the same method as the login page (setting a variable \$error to contain the message, then displaying it using appropriate styling).

Using the pseudocode developed for registering users in [2.5.3](#), I built the validation for this process. Using the function mysqli\_real\_escape\_string, every input is sanitised to prevent

SQL injection. The code will check that something has been entered for all fields (presence check). Then, I developed code to validate the first name. The name must be 2-50 characters, have no special characters (except hyphens) and no numbers. I developed the length check using `strlen()`, and used `preg_match` to conduct the format checks.

```
// check first name validity - Name must be 2-50 chars
$validLength_fname = strlen($firstName) >= 2 && strlen($firstName) <= 50;
// no special chars (only lowercase, uppercase, hyphens, and spaces)
$characterSetValid_fname = preg_match("/^([a-zA-Z\-\ ])+$/", $firstName);
// No numbers
$noNumbers_fname = !preg_match("/[0-9]/", $firstName);

if ($validLength_fname and $characterSetValid_fname and $noNumbers_fname) {
```

Each variable above is boolean, and if all are true the first name is valid. The function `preg_match` uses specific patterns and syntax to allow me to develop specific format checks. In the first case where this function is used, the syntax `[a-zA-z\-\ ]+` specifies that every character must be in the range a-z, A-Z, or must be a hyphen or a space. The hyphen (-) followed by space specifies these characters are allowed. If any character does not match this pattern, the name is invalid so the function returns false. When checking that there are no numbers, the function will return true if it finds one character in the range 0-9. The exclamation marks acts as a NOT gate for this variable, so the variable stores the opposite of the function. If no numbers are found, `preg_match` returns false and the variable stores true. If everything is valid, the same process is repeated for the last name. If either of these are invalid, an appropriate error message is displayed. The next validation check uses `filter_var` to validate emails according to RFC 822 regulations.

If all validation has been passed, the program must check if the email address is already stored in Students or Supervisors. If it is, an error message is displayed.

```
//check if user already has a login
$checkStudents="SELECT * FROM Students WHERE Email = '$email'";
$checkSupervisors="SELECT * FROM Supervisors WHERE Email = '$email'";
$studentResult=$conn->query($checkStudents);
$supervisorResult=$conn->query($checkSupervisors);
```

If the email has been found, one of the 2 results will have a number of rows  $> 0$ . This will return an error message specifying where the email has been registered, for clarity. If no email is found, the algorithm continues. The password is first verified, by checking that both entries of the password are equal. Then, it is validated, once again using `preg_match`.

```
//check password constraints
$uppercase = preg_match('@[A-Z]@', $password);
$lowercase = preg_match('@[a-z]@', $password);
$number = preg_match('@[0-9]@', $password);
if (!$uppercase or !$lowercase or !$number or strlen($password) < 8) {
    $error="Password must be at least 8 characters long and include at least one uppercase letter, one lowercase letter, and one number.";
```

The format checks above are the same as designed in section [2.4](#). \$uppercase is true if the password contains an uppercase letter (ie. in the range A-Z). \$lowercase is true if the password contains a lowercase letter (a-z). \$number is true if the password has a number (ie. a character in the range 0-9). If any of these are false, or the length of the password is below 8, an error is displayed.

```
// hash password for security
$hashedPassword=password_hash($password,PASSWORD_DEFAULT);
// register into database
$sql="INSERT INTO Students (FirstName,LastName,Email,Password,ParentEmail)
VALUES ('$firstName','$lastName','$email','$hashedPassword','$parentEmail')";
if ($conn->query($sql) === TRUE){
    header("Location: login.php?registered=1");
    exit();
} else {
    $error= "Error with database: " . $sql . "<br>" . $conn->error;
}
```

The password\_hash function will hash the password, making it secure. As hashing is a one way process, no one can obtain the password from its hash, protecting this data even if the database is hacked. The PASSWORD\_DEFAULT currently means that the function will use the bcrypt algorithm for hashing, as it is strong and reliable. If a more appropriate hashing algorithm is developed, the algorithm used here will update. Therefore, using PASSWORD\_DEFAULT ensures that the hashing algorithm is always strong, so the system will always be secure in the future. This makes my system more durable and future-proof.

SQL is used to insert all the values into the table, creating a new record. Once successfully submitted, the page will redirect to the login page. While developing this feature, I decided that the login page should display some message to show that the user has been registered. Otherwise, the user may be confused, possibly thinking that they clicked the wrong button. By displaying a message, the user can feel at ease with the software. Therefore, the variable registered=1 is attached in the header() function. This variable will be accessible at login.php.

```

<!-- if user has just registered -->
<?php if (isset($_GET['registered']) and $_GET['registered']==1) { ?>
    <div class="alert alert-success">
        | Registration successful!
    </div>
<?php } ?>

```

I added this code in login.php . Although not in my design, I believe that this feature allows the system to become more user friendly, and develops a professional look in the system. The same code was then copied into register-supervisor.php , but all inputs and validation of the parent email was removed. I then adjusted the file so that it inserted the data into Supervisors, not Students.

**Register a Music Teacher / Accompanist**

First name  
  
Names must be 2-50 characters, with no numbers or special characters (except hyphens).

Last name

Email address

Password  
  
Password must be at least 8 characters long and include at least one uppercase letter, one lowercase letter and one number.

Confirm Password

**Register**

If you have an account, login [here](#)

The interface for register-supervisor.php is shown above. I have intentionally chosen ‘Register a Music Teacher / Accompanist’ instead of ‘Register a Supervisor’, as this term is relatively specific to this project, and is quite likely to confuse people.

### 3.3.2 Unit testing

As there is lots of different validation being used across 2 separate pages, there are lots of tests to carry out. It is crucial that I test all parts of this code, testing the full functionality of this unit gives me confidence that it can be integrated into the full solution with no errors. All results of tests are given underneath their table.

#### REGISTER-STUDENT.PHP

Purpose of test	Test data	Expected Result	Successful?
-----------------	-----------	-----------------	-------------

Detects if nothing has been entered	NULL	"You must enter data for all fields"	Yes
-------------------------------------	------	--------------------------------------	-----

## Register a Student

You must enter data for all fields

First name

Names must be 2-50 characters, with no numbers or special characters (except hyphens).

Last name

Email address

Parent email address

Password

Password must be at least 8 characters long and include at least one uppercase letter, one lowercase letter and one number.

Confirm Password

If you have an account, login [here](#)

Purpose of test	Test data	Expected Result	Successful?
Detects an invalid first name due to numbers	First name = '123' Last name = 'desai' Email = 'test@example.com' Parent email = 'parent@email.com' password= 'Password1' Confirm password = 'Password1'	'Both names must be 2-50 characters, with no numbers or special characters (except hyphens)'	Yes

## Register a Student

Both names must be 2-50 characters, with no numbers or special characters (except hyphens)

Purpose of test	Test data	Expected Result	Successful?
Detects an invalid first name due to short length (1 character)	First name = 'a' All other fields same as above	'Both names must be 2-50 characters, with no numbers or special characters (except hyphens)'	Yes

## Register a Student

Both names must be 2-50 characters, with no numbers or special characters (except hyphens)

Purpose of test	Test data	Expected Result	Successful?
Detects an invalid first name due to long length (51 characters)	First name = 'ABCDEFGHIJKLMNO PQRSTUWXYZABC DEFGHIJKLMNOPQR STUVWXY' All other fields same as above	'Both names must be 2-50 characters, with no numbers or special characters (except hyphens)'	Yes

Both names must be 2-50 characters, with no numbers or special characters (except hyphens)

Purpose of test	Test data	Expected Result	Successful?
Detects an invalid first name due to special characters	First name = 'test@' All other fields same as above	'Both names must be 2-50 characters, with no numbers or special characters (except hyphens)'	Yes

Both names must be 2-50 characters, with no numbers or special characters (except hyphens)

Purpose of test	Test data	Expected Result	Successful?

Detects an invalid last name due to numbers	First name = 'rohan' Last name = '789'  Email = 'test@example.com' Parent email = 'parent@email.com' password= 'Password1' Confirm password = 'Password1'	'Both names must be 2-50 characters, with no numbers or special characters (except hyphens)'	Yes
---	--	--	-----

Both names must be 2-50 characters, with no numbers or special characters (except hyphens)

Purpose of test	Test data	Expected Result	Successful?
Detects an invalid last name due to short length (1 character)	Last name = 'r'  All other fields same as above	'Both names must be 2-50 characters, with no numbers or special characters (except hyphens)'	Yes

Both names must be 2-50 characters, with no numbers or special characters (except hyphens)

Purpose of test	Test data	Expected Result	Successful?
Detects an invalid last name due to long length (51 characters) (Boundary data)	Last name = 'abcdefghijklmnopqrstuvwxyzabcdefghijklmnopqrstuvwxyzabcdefghijklmnopqrstuvwxyzabcdefghijklmnopqrstuvwxyz'  All other fields same as above	'Both names must be 2-50 characters, with no numbers or special characters (except hyphens)'	Yes

Both names must be 2-50 characters, with no numbers or special characters (except hyphens)

Purpose of test	Test data	Expected Result	Successful?

Detects an invalid last name due to special characters	Last name = ‘test£’  All other fields same as above	‘Both names must be 2-50 characters, with no numbers or special characters (except hyphens)’	Yes
--	---	--	-----

Both names must be 2-50 characters, with no numbers or special characters (except hyphens)

Purpose of test	Test data	Expected Result	Successful?
Detects an invalid student email	First name = ‘Rohan’  Last name = ‘Desai’  Email = ‘a@b’  Parent email = ‘parent@email.com’ password= ‘Password1’ Confirm password = ‘Password1’	‘Invalid email / parent email’	Yes

Invalid email / parent email

Purpose of test	Test data	Expected Result	Successful?
Detects an invalid parent email	Email = ‘student@email.com’  Parent email = ‘b@c’  All other fields same as above	‘Invalid email / parent email’	Yes

Invalid email / parent email

Purpose of test	Test data	Expected Result	Successful?
Detects if an email exists in	First name = ‘Stu’ Last name = ‘Dent’	‘Email already registered as a	No

Students	Email = <a href="mailto:student@email.com">student@email.com</a>  Parent email = <a href="mailto:parent@email.com">parent@email.com</a> Password = 'Student123' Confirm password = 'Student123'  <b>This is already in the Students table (see <a href="#">3.2.1</a>)</b>	student. Please log in.'	
----------	---	--------------------------	--

Email already registered as a music teacher / accompanist. Please log in

The error message displayed was incorrect, saying that the email already exists as a supervisor's login. However, this is a student login. The code has correctly recognised that this email is already in one of the tables, which allowed me to look at only a small section of the code for errors. I soon realised that I had mixed up \$studentResult and \$supervisorResult.

```
if ($supervisorResult->num_rows>0) {
    // email found in Students table
    $error = "Email already registered as a student. Please log in";
} else {

    if($studentResult->num_rows>0) {
        // email found in Supervisors table
        $error = "Email already registered as a music teacher / accompanist. Please log in";
    } else {
```

By switching the 2 results, I ensured that each error gave the appropriate message. I made the same change on register-supervisor.php

```
if ($studentResult->num_rows>0) {
    // email found in Students table
    $error = "Email already registered as a student. Please log in";
} else {

    if($supervisorResult->num_rows>0) {
        // email found in Supervisors table
        $error = "Email already registered as a music teacher / accompanist. Please log in";
    } else {
```

I tested this again, to check if anything had changed.

Purpose of test	Test data	Expected Result	Successful?
Detects if an email exists in Students	First name = 'Stu' Last name = 'Dent'	'Email already registered as a student. Please log in'	Yes

Email = <a href="mailto:student@email.com">'student@email.com'</a>  Parent email = <a href="mailto:parent@email.com">'parent@email.com'</a> Password = 'Student123' Confirm password = 'Student123'	in.'	
---	------	--

Email already registered as a student. Please log in

Purpose of test	Test data	Expected Result	Successful?
Detects if an email exists in Supervisors	First name = 'John' Last name = 'Doe'  Email = 'john.doe@email.com'  Parent email = <a href="mailto:parent@email.com">'parent@email.com'</a> Password = 'Johndoe1' Confirm password = 'Johndoe1'  <b>This email is already in Supervisors table (see 3.2.1)</b>	'Email already registered as a music teacher / accompanist. Please log in.'	Yes

Email already registered as a music teacher / accompanist. Please log in

Purpose of test	Test data	Expected Result	Successful?
Detects if passwords do not match	First name = 'rohan' Last name = 'desai' Email = 'test@example.com' Parent email = <a href="mailto:parent@email.com">'parent@email.com'</a>  password= 'Password1' Confirm password = 'Password2'	'Passwords do not match'	Yes

Passwords do not match

Purpose of test	Test data	Expected Result	Successful?
Detects an invalid password due to no uppercase letters	Password = 'password1' Confirm password = 'password1'  All other fields same as above	"The password must be at least 8 characters long and include at least one uppercase letter, one lowercase letter, and one number."	Yes

The password must be at least 8 characters long and include at least one uppercase letter, one lowercase letter, and one number.

Purpose of test	Test data	Expected Result	Successful?
Detects an invalid password due to no lowercase letters	Password = 'PASSWORD1' Confirm password = 'PASSWORD1'  All other fields same as above	"The password must be at least 8 characters long and include at least one uppercase letter, one lowercase letter, and one number."	Yes

The password must be at least 8 characters long and include at least one uppercase letter, one lowercase letter, and one number.

Purpose of test	Test data	Expected Result	Successful?
Detects an invalid password due to no numbers	Password = 'Password' Confirm password = 'Password'  All other fields same as above	"The password must be at least 8 characters long and include at least one uppercase letter, one lowercase letter, and one number."	Yes

The password must be at least 8 characters long and include at least one uppercase letter, one lowercase letter, and one number.

Purpose of test	Test data	Expected Result	Successful?
Detects an invalid password due to short length (4 characters)	Password = 'Pas1' Confirm password = 'Pas1'  All other fields same as above	"The password must be at least 8 characters long and include at least one uppercase letter, one lowercase letter, and one number."	Yes

The password must be at least 8 characters long and include at least one uppercase letter, one lowercase letter, and one number.

Purpose of test	Test data	Expected Result	Successful?
Boundary data: detects invalid password due to short length (7 characters)	Password = 'Tester1' Confirm password = 'Tester1'  All other fields same as above	"The password must be at least 8 characters long and include at least one uppercase letter, one lowercase letter, and one number."	Yes

The password must be at least 8 characters long and include at least one uppercase letter, one lowercase letter, and one number.

Purpose of test	Test data	Expected Result	Successful?
Accepts valid details	First name = "John" Last name = "Smith" Email = 'j.smith@email.com' Parent email = 'parent.smith@email.com' Password = 'Johnsmith123' Confirm password = 'Johnsmith123'	Redirected to login page with message 'Registration successful'  Also, the Students table shows the new record.	Yes

--	--	--	--

## Login

Registration successful!

Email address

Password

Don't have an account? [Register here](#)

StudentID	FirstName	LastName	Email	Password	ParentEmail
32	John	Smith	j.smith@email.com	\$2y\$10\$qIIQ32jSEiliYKaI	parent.smith@email.com

This test is successful, and all of the data has been stored correctly in the table, with StudentID automatically created upon registration. In order to be successfully registered, the data must pass through all validation checks. Therefore, this test shows that all validation and verification works correctly with valid data.

Purpose of test	Test data	Expected Result	Successful?
Accepts valid details with boundary data for the first name (2 characters), last name (50 characters) and password (8 characters).	First name = "Jo" Last name = "abcdefghijklmnopqrstuvwxyzabcdefghijklmnopqrstuvwxyz" Email = 'testing@email.com' Parent email = 'parent.test@email.com' Password = 'Passw0rd' Confirm password = 'Passw0rd'	Redirected to login page with message 'Registration successful'  Also, the Students table shows the new record.	Yes
Testing the extremes of this feature and its validation			

**Login**

Registration successful!

Email address	<input type="text"/>
Password	<input type="password"/>
<input style="background-color: #007bff; color: white; padding: 5px; border-radius: 5px; border: none; font-weight: bold; width: fit-content; margin: auto;" type="button" value="Login"/>	
Don't have an account? <a href="#">Register here</a>	

StudentID	1	FirstName	LastName	Email	Password	ParentEmail
33	Jo	abcdefghijklmno	pqrstuvwxyz	testing@email.com	\$2y\$10\$cNC5jbSn	parent.test@email.com

The tests above show that students can register themselves using this page, and the code is error-free. All validation is working correctly, and database management is also successful.

### REGISTER-SUPERVISOR.PHP

The **exact** validation has been used in the page for registering supervisors, register-supervisor.php . Therefore, I will not document the exact same tests again, as the results are all the same. The only difference is that supervisors do not enter a parent email, so any input and validation surrounding that is removed. Also, data is entered into Supervisors, not Students, using the following query:

```
// insert into database
$sql="INSERT INTO Supervisors (FirstName,LastName,Email,Password)
VALUES ('$firstName','$lastName','$email','$hashedPassword');
```

I will not test the validation checks of this page, as they are the exact same as register-student.php . However, I will test this new query by making a valid registration.

Purpose of test	Test data	Expected Result	Successful?
Accepts valid details to register a new supervisor	First name = "Benjamin" Last name = "Brown" Email = 'brown@email.com' Password = 'Brown123'	Redirected to login page with message 'Registration successful'	Yes

	Confirm password = 'Brown123'	Also, the Supervisors table shows the new record.	
--	----------------------------------	---	--

The screenshot shows a login page with a green success message box at the top containing the text "Registration successful!". Below it is a form with fields for "Email address" and "Password", each with a corresponding input field. A blue "Login" button is positioned below the password field. At the bottom of the page, there is a link "Don't have an account? [Register here](#)".

SupervisorID	FirstName	LastName	Email	Password
19	Benjamin	Brown	brown@email.com	\$2y\$10\$OrEx2crWiCk3

I am now confident that both pages have been tested fully, to test the full range of functionality of this feature.

### 3.3.3 System test of login and registration

I am conducting a short system test, testing the process of registering a new user and logging them into the system. This will combine the 2 modules from stage 3.2 and 3.3, as well as the database setup from 3.1. There will be 2 main tests: one for students and one for supervisors. By combining these 2 modules and testing them as one, I can examine how this prototype of the full solution will work. This is important as it allows me to understand how the modules behave when integrated into the full system, and correct any unexpected results. The tests will involve registering and immediately logging in to the system, which will be carried out by all users when this project is implemented. Therefore, it is useful to test this common sequence of events, and document how the modules work together.

#### *Register and login - Student*

Purpose of test	Test data	Expected Result	Successful?
-----------------	-----------	-----------------	-------------

Testing registration of a student	First name = "Tim" Last name = "Jones" Email = "tim@email.com" Parent email = "tim.parent@email.com" Password = "TimJones1" Confirm password = "TimJones1"	Redirected to login page with message 'Registration successful'  Also, the Students table shows the new record.	Yes
-----------------------------------	---	---	-----

## Register a Student

First name  
  
Names must be 2-50 characters, with no numbers or special characters (except hyphens).

Last name

Email address

Parent email address

Password  
  
Password must be at least 8 characters long and include at least one uppercase letter, one lowercase letter and one number.

Confirm Password  


**Register**

If you have an account, login [here](#)

## Login

Registration successful!

StudentID	FirstName	LastName	Email	Password	ParentEmail
34	Tim	Jones	tim@email.com	\$2y\$10\$n5K2OFBlrn	tim.parent@email.com

Purpose of test	Test data	Expected Result	Successful?
Testing login of this new student	Email = "tim@email.com" Password = "TimJones1"	Redirected to dashboard	Yes

Email address	<input type="text" value="tim@email.com"/>
Password	<input type="password" value="....."/>
<input type="button" value="Login"/>	

SoundBro. [Dashboard](#) [Logout](#)

## Dashboard

Welcome, Tim!

Rohan Desai 2023  
King Edward VI Camp Hill School for Boys

This test shows that both modules, when combined, work well together with no errors. I will repeat this test for supervisors, to ensure that this process works smoothly for all users, not just students.

### *Register and login - Supervisor*

Purpose of test	Test data	Expected Result	Successful?
Testing registration of a supervisor	First name = "Jack" Last name = "Felton" Email = "felton@email.com" Password = 'JFelton1' Confirm password = 'JFelton1'	Redirected to login page with message 'Registration successful'  Also, the Supervisors table shows the new record.	Yes

## Register a Music Teacher / Accompanist

First name

Names must be 2-50 characters, with no numbers or special characters (except hyphens).

Last name

Email address

Password

Password must be at least 8 characters long and include at least one uppercase letter, one lowercase letter and one number.

Confirm Password

If you have an account, login [here](#)

## Login

Registration successful!

SupervisorID	FirstName	LastName	Email	Password
20	Jack	Felton	felton@email.com	\$2y\$10\$vTJsCdckH4y6hJoV

Purpose of test	Test data	Expected Result	Successful?
Testing login of this new supervisor	Email = "felton@email.com" Password = "JFelton1"	Redirected to dashboard	Yes

Email address

Password

The screenshot shows the SoundBro dashboard. At the top, there are links for 'Dashboard' and 'Logout'. Below that, the word 'Dashboard' is centered in a large font. Underneath it, the text 'Welcome, Jack!' is displayed. A grey horizontal bar contains a small icon of two overlapping circles and the text 'Rohan Desai 2023 King Edward VI Camp Hill School for Boys'.

This test was also successful. This short system test has shown that the modules 3.1, 3.2, and 3.3 have integrated smoothly into the full solution, with no errors. This shows that the prototype of my solution is error free, allowing me to confidently move onto the next stage.

### *Stakeholder feedback - Darius Maleki-Tooserkani*

In order to ensure that this prototype of my solution is being implemented correctly, I checked with one of my stakeholders, Darius. We have previously discussed the design of the web page (see [2.3](#)). By showing this prototype to him, I can confirm that the development of the project is in accordance with stakeholder requirements. Darius' feedback will allow me to understand what he likes and dislikes about the system, so that I can make changes to improve its features. I showed my prototype to him in person, he navigated through all pages and tried registering and logging in. We discussed the possible improvements to the system in person. However, I also got him to send me an email detailing all of the feedback that he gave.

The screenshot shows an email interface. The recipient is 'Darius Maleki-Tooserkani' with a note 'to me'. The timestamp is '11:39 (1 hour ago)'. The message body starts with 'Hi Rohan,' followed by 'Here is my feedback from earlier today:'. The feedback text continues: 'The system looks good overall, the interface looks very similar to what we agreed on before, which is nice. I especially like the icons, it makes the system look more professional. During registration, I think you could explain why you are collecting a parent's email, like a small message under the box. I wouldn't want to enter it otherwise because I wouldn't trust it as much. Also, you should probably send the user an email once they've registered.' The message concludes with 'Thanks, Darius.'

Most of the feedback was positive, but Darius suggested 2 changes to the system. One change was to send the user an email upon account creation. This feature was originally in the designed solution, but was later removed in section 2.5.1. It can also be seen as success criteria L6, or structure diagram box 1.2.1.2. I would also like to implement this feature, however there are several reasons why it has been removed from the final project. These

reasons can be seen in section [2.5.1](#). As a result of Darius' comment, I hope to develop this feature in the near future. However, it will not be developed during the timeframe of this project.

The other criticism from Darius was the lack of explanation surrounding the parent email address. His suggestion was to include an on screen message during registration, to 'explain why you are collecting a parent's email'. This will inform the user about why the data is being collected, so they trust the system more and feel more comfortable to enter this data. I completely agree with this suggestion, and decided to implement this feature.

```
<!-- enter parent email -->
<div class="form-group">
    <label for="parentEmail">Parent email address </label>
    <input type="email" class="form-control" id="parentEmail" name="parentEmail">
    <small class="text-muted">
        We'll only send you and your parent an email when the schedule is created.
    </small>
</div>
```

Using the `<small>` tag, I added a quick explanation to what the email addresses are used for. Most students will be wary of giving out their email address, as they can get lots of emails for marketing purposes etc. This message clarifies the purpose of the email, and mitigates any concerns.

A screenshot of a web form. At the top, there is a label "Parent email address" followed by an input field. Below the input field is a `<small>` tag containing the text "We'll only send you and your parent an email when the schedule is created."

I made a similar change in `register-supervisor.php` as well. This message makes users feel more comfortable with entering data, and prevents any confusion. This makes the overall system easier to use, providing a better experience for the user. I showed this result to Darius, who was very happy with the change. We agreed that it clearly allows users to trust the solution more.

### 3.4.3 Review of stage 3

#### *Stage 3 summary*

- Developed `register.php`, to direct students and supervisors to separate pages
- Developed `register-student.php` to register students
- Developed `register-supervisor.php` to register supervisors

#### *Testing carried out*

- Tested validation of first name, last name, email, parent email, password
- Tested verification of confirming a password when entering twice

- Tested process of entering valid user data into the database
- Conducted system test of 3.1, 3.2 and 3.3

All tests were successful in the end

### *Success criteria met*

The successful development of this stage has allowed me to meet success criteria L3, as users can now register themselves using register-student.php or register-supervisor.php.

Additionally, success criteria L4 has been achieved, as every single field goes through a careful process of sanitising, validating and sometimes verifying the inputs. This ensures that the data is safe, sensible and more likely to be correct.

### *Changes to the design*

I have added to my design by including a message upon successful registration. If registered successfully, the user is taken to the login page and a message is displayed: "Registration successful!". This message is in green to show success, and it helps the user to understand the inner processes of this software. I have also included more helpful messages to prevent confusion and make the system easier to use.

### *Review of the project so far*

The current system involves the databases, which are working very well so far, the registration page and the login page. My system test has shown that these 3 modules can be integrated into my solution to create a fully functioning prototype of my product. I am now ready to move onto the next stage of the project, developing the scheduling functionality. My directory structure has been updated and is shown below. The 3 pages created in this section are displayed in red.

```

soundbro/
|--- index.php
|--- login.php
|--- register.php
|--- register-student.php
|--- register-supervisor.php
|--- dashboard.php
|--- logout.php
|--- styles.css
|--- includes/
    |--- db-connect.php
    |--- navbar.php
    |--- footer.html
|--- logo/
    |--- full.png (main logo)
    |--- favicon3.png

```

## 3.4 *nextBreak* function

The next few stages will concern the most important part of this project: the scheduling process. To develop this feature, I will implement all algorithms detailed in section [2.5.2](#). Together, they will form the schedule function, a single function that takes the validated data in the database to produce a schedule and display it on the screen. The creation of this feature inspired the entire project, and my client Ms Pearcey specified in section [2.5.1](#) that the development of this feature has the highest priority. Please note that these stages will not include much validation, as all key validation occurs in earlier parts of the scheduling process, when data is entered into the database (structure diagram 1). These stages will use data that has already been validated, any invalid data would return an error that prevents this function from being accessible. Initially, I will develop each module completely separately from each other. This makes my solution much more modular, allowing me to test the full range of functionality in each module. Finally, I will combine the modules together and conduct a system test, before integrating it into the prototype of the solution from the previous stage. Therefore, the pages developed will not yet be linked to my login and registration features, but will exist in the same directory. They will also be displayed in the same style as previous pages, for consistency.

As section 2.5.2 shows, the schedule function calls many other functions. These smaller subroutines must be developed before the schedule function can be developed. This section will involve developing the function ‘nextBreak’. This function is used repeatedly, and is crucial to the success of the solution.

### 3.4.1 Developing

Using the flowchart from section 2.5.2 I was able to develop the function below. I created this function in a separate file, ‘schedule.php’. This file will be used to develop the schedule function. It is in the same style as previous pages, but is completely modular and self-contained, as it does not interact with the login and registration functionality yet. This allows me to conduct accurate, comprehensive unit testing on each module, as specified in section [2.6](#). The function has one parameter, numCompletedBreaks. There is further clarification on this variable in section [2.4.2](#).

```

function nextBreak($numCompletedBreaks){
    $error = "";
    // validate input
    if ($numCompletedBreaks >= 0 and $numCompletedBreaks <= 3) {
        // set next break time based on number of breaks completed
        switch($numCompletedBreaks){
            case 0:
                $breaktime = new DateTime('10:45');
            case 1:
                $breaktime = new DateTime('12:30');
            case 2:
                $breaktime = new DateTime('15:15');
            case 3:
                $breaktime = new DateTime('17:00');
        }
    } else {
        $error = "ERROR - Invalid number of completed breaks. ";
        $breaktime = new DateTime('23:00');
    }
    return [$breaktime, $error];
}

```

I could have used a series of if/else statements to decide what \$breaktime should contain. However, I deemed a switch/case statement to be much more appropriate, as I am only evaluating a certain number of cases.

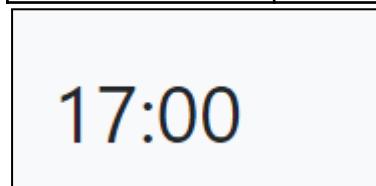
I have changed this function from my design, to become more robust and handle all types of inputs. Firstly, it validates the input to ensure it is between 0 and 3. If not, there is an error, as this variable should never be outside this range. Therefore, \$error will be returned, containing the error message. In this case, I have returned \$breaktime as 23:00 to prevent a fatal error in the code. When \$error is returned to the schedule function, it will cause the while loop to stop the scheduling process, and the error message will be displayed. This prevents any further logic errors from occurring. The variable \$error will store **all** errors found in the code, so when returned to the calling procedure, this message would be stored along with other existing messages.

If the input is valid, the function sets the (ideal) time of the next break. I have returned \$breaktime as an object of the DateTime class, as this is the best way to store the time. The object will later be compared to the 'time' object, which also uses DateTime. As they are both using the same class, certain methods can be used to compare them easily. The benefit of including this code as a separate function allows me to use it repeatedly during the scheduling process. It also allows me to easily edit these times. By changing the value of \$breaktime in this function, the entire process will consistently reflect the changes. This makes the program easier to maintain.

### 3.4.2 Unit testing

If an error is found, it will be displayed in red on the screen. Either way, I have used ‘echo’ to output the returned break time to the screen. This allows me to see its outputs clearly.

Purpose of test	Test data	Expected Result	Successful?
Testing the function when no breaks have been set (valid data)	numCompletedBreaks=0	10:45 outputted on screen	No



This test failed, so I looked closer at the function. Clearly, the error is "", otherwise an error message would be displayed. Therefore, the if statement is working fine. The error must therefore be in the switch case statement. After some research, I learned that each case must end in ‘break;’ or it will not start the next case. I confirmed that this was the issue by finding that the output was 17:00 when numCompletedBreaks is 0,1,2, or 3. Therefore, the variable must be overwriting itself each time. To fix this problem, I simply included ‘break;’ at the end of each case, as shown below.

```
case 0:  
    $breaktime = new DateTime('10:45');  
    break;  
case 1:  
    $breaktime = new DateTime('12:30');  
    break;  
case 2:  
    $breaktime = new DateTime('15:15');  
    break;  
case 3:  
    $breaktime = new DateTime('17:00');  
    break;
```

Then, I ran the same test again.

Purpose of test	Test data	Expected Result	Successful?
Testing the function when no breaks have been set (valid data)	numCompletedBreaks=0	10:45 outputted on screen	Yes

--	--	--	--

10:45

Purpose of test	Test data	Expected Result	Successful?
Testing the function when 1 break has been set (valid data)	numCompletedBreaks=1	12:30 outputted on screen	Yes

12:30

Purpose of test	Test data	Expected Result	Successful?
Testing the function when 2 breaks have been set (valid data)	numCompletedBreaks=2	15:15 outputted on screen	Yes

15:15

Purpose of test	Test data	Expected Result	Successful?
Testing the function when 3 breaks have been set (valid boundary data)	numCompletedBreaks=3	17:00 outputted on screen	Yes

17:00

The success of all tests above show that the function handles valid data correctly. The inputs and outputs exactly match the flowchart created in section 2.5.2. The next tests have been included to test invalid data. Currently, if \$error is not an empty string, it is outputted. This exact comparison will be used in the while loop in the schedule function. If \$error is not an empty string, the while loop will end and the error is displayed.

```
<!-- display if error is found -->
<?php if ($error!="") { ?>
    <div class="alert alert-danger">
        <?php echo $error; ?>
    </div>
<?php } ?>
```

By using similar logic to the full system, I can catch any errors that may occur in the full system here. This makes it easier to integrate each module into the system.

Purpose of test	Test data	Expected Result	Successful?
Testing the function when 4 breaks have been set (invalid boundary data)	numCompletedBreaks=4	23:00 outputted on screen  Error message displayed “ERROR - Invalid number of completed breaks.”	Yes

23:00

ERROR - Invalid number of completed breaks.

Purpose of test	Test data	Expected Result	Successful?
Testing the function with	numCompletedBreaks= -1	23:00 outputted on screen	Yes

invalid data		Error message displayed “ERROR - Invalid number of completed breaks.”	
--------------	--	---	--

23:00

ERROR - Invalid number of completed breaks.

Purpose of test	Test data	Expected Result	Successful?
Testing the function with invalid data	numCompletedBreaks= ‘three’	23:00 outputted on screen  Error message displayed “ERROR - Invalid number of completed breaks.”	Yes

23:00

ERROR - Invalid number of completed breaks.

### 3.4.3 Review of stage 4

#### *Stage 4 summary*

- Developed nextBreak function
- Developed more validation to make the function more robust

#### *Testing carried out*

- Tested with valid data
- Tested with boundary data (valid and invalid)
- Tested with invalid / erroneous data

All tests were successful in the end

### *Success criteria met*

No success criteria have explicitly been met in this section. However, it allows criteria S1-11 to be met in later stages. This function will be used to help the schedule function to decide whether to schedule an exam or a break, depending on what is most efficient (S3) within the regulations. Therefore, this function also enables the schedule to conform to TCL's and Ms Pearcey's regulations (S1). It also allows the squeeze function to decide which exam should be scheduled so minimal time is wasted (S2).

### *Changes to the design*

The designed function has been implemented in this stage. However, it has been improved so it is slightly more complex than my design originally specified. Now, this function will validate inputs more, and return an error message as a string. The implementation of the schedule function will be updated to store this error message and process it appropriately.

### *Review of the project so far*

The development of this function is the first step towards a successful scheduling function. The next few stages will keep developing functions like this, which together can be used in the larger schedule function. Here is the current directory structure. The only change from the last stage is the file 'schedule.php', which is shown in red.

```
soundbro/
|--- index.php
|--- login.php
|--- register.php
|--- register-student.php
|--- register-supervisor.php
|--- dashboard.php
|--- schedule.php
|--- logout.php
|--- styles.css
|--- includes/
    |--- db-connect.php
    |--- navbar.php
    |--- footer.html
|--- logo/
    |--- full.png (main logo)
    |--- favicon3.png
```

## 3.5 meetingRegulations function

### 3.5.1 Developing

This function will take a series of variables related to the current status of the schedule to decide whether or not it is meeting TCL regulations. These regulations are strict so the function will only return true or false; there is no leniency. The function has 4 parameters:

- **time**
  - Object of DateTime class
  - Stores the current time of the schedule (alternatively, it stores the end time of the most recent exam)
- **dayExamTime**
  - Integer variable
  - Stores the total number of minutes of exams so far
- **sessionTime**
  - Integer variable
  - Stores the number of minutes of the current session (minutes since the last break)
- **numCompletedBreaks**
  - Integer variable
  - Stores the number of breaks scheduled so far

All of these variables have no user input (they are initialised earlier in the scheduling process), so need no validation or sanitisation. All of these variables have been defined more rigorously in [2.4.2](#). This function will help to decide whether or not to schedule an exam.

The flowchart for this function in section 2.5.2 shows that the entire function can be simplified to one if statement, if the condition is true, the function returns true. If not, the function returns false. As the value of the logical condition is returned, the entire function can be simplified into one line! This line would return quite a lengthy logical condition. However, this is not a good structure, as it is not easy to understand. Therefore, each of the 4 logical comparisons have been defined as a boolean variable, and these 4 variables are combined in the ‘return’ statement.

```
return $timeValid and $lunchValid and $sessionValid and $dayExamValid ;
```

Each of these 4 variables relate to the exam regulations from section [1.1.2](#), more specifically from figure 1.1.2a, which is repeated below.

## MUSIC TIMETABLES

A standard start time is anywhere between 9:00 and 10:00am. The examiner will arrive 30 minutes before the first exam.

An examiner may work for no more than 2 hours without a 15 minute break. A one hour lunch break should be timetabled after no more than 3.5 hours. The maximum exam time allowed per day (not including breaks) is 6.5 hours. For a full day of exams, a one hour lunch break and 15 minute breaks in mid-morning and mid-afternoon should be included. This is illustrated in the following example:

### SAMPLE TIMETABLE

EXAMS	BREAKS
09:00-11:00	11:00-11:15
11:15-12:45	12:45-13:45 (lunch)
13:45-15:15	15:15-15:30
15:30-17:00	

The total number of hours an examiner spends at the centre in one day should not exceed 8 hours. This includes both examining time and breaks. Deviations from this standard must be cleared with Trinity no less than four weeks in advance of the exam date.

(repeated) *Figure 1.1.2a: Rules for timetabling exams*

“The total number of hours an examiner spends at the centre in one day should not exceed 8 hours”. As the start time for an exam in my school is always 9am (according to Ms Pearcey), the latest time an exam can end is 5pm. Therefore, if the time is past 5pm, the schedule is currently breaking regulations. This logic is contained in \$timeValid.

“A one hour lunch break should be scheduled after no more than 3.5hours.” This statement excludes the 15 minute morning break (I checked this with Ms Pearcey). Again, exams start at 9am, so 3.5hours later (excluding the morning break) is 12:45. \$lunchValid is true if this condition is met. If a lunch break has not been scheduled, and the time is later than 12:45, this variable is false as it is breaking regulations.

“An examiner may work for no more than 2 hours without a 15 minute break”. If the session time exceeds 120 minutes, regulations are broken and \$sessionValid is false. If not, this variable is true.

“The maximum exam time allowed per day (not including breaks) is 6.5 hours.” 6.5 hours is 390 minutes, so if dayExamTime exceeds this value then \$dayExamValid is false.

If and only if all variables are true, then regulations are being met. If any variables are false, regulations are being broken. Using ‘and’ with all of these variables in the return statement ensures that the function meets these requirements. The full function is shown below.

```

function meetingRegulations($time,$dayExamTime,$sessionTime,$numCompletedBreaks){
    $latestTime = new DateTime('17:00');
    $latestLunchTime = new DateTime('12:45');

    // decide if time is valid
    $timeValid = $time->format('H:i') <= $latestTime->format('H:i');
    // decide if a lunch break must be scheduled or not
    $lunchValid = ($numCompletedBreaks != 1) OR ($time->format('H:i') <= $latestLunchTime->format('H:i'));
    // decide if the current session is too long
    $sessionValid = $sessionTime <= 120;
    // decide if too many exams have been scheduled
    $dayExamValid = $dayExamTime <= 390;

    return $timeValid and $lunchValid and $sessionValid and $dayExamValid ;
}

```

As the TCL rules often change, it is important that it is easy to update the system to reflect these changes. Creating this function makes all changes consistent throughout the scheduling process, as editing this function will edit all occurrences of it. By creating \$latestTime and \$latestLunchTime, the function becomes more understandable and it is easier to adjust these times.

If numCompletedBreaks is 0, a lunch break has not been scheduled. However, it also means that a morning break has not been scheduled (at 10:45). Therefore, if this variable is 0, it is not necessary to check the time, as we already know it is not anywhere near 12:45. If numCompletedBreaks is 1, a lunch break should be scheduled next. Therefore, we must ensure it is before 12:45, otherwise regulations are being broken and \$lunchValid is false, causing the function to return false. The variables inputted into this function will essentially detail the proposed exam to be scheduled next, so if the function returns false the proposed exam will be rejected, it will schedule a break (or try to squeeze in 1 more exam).

### 3.5.2 Unit testing

I tested this function with a wide variety of valid, invalid and boundary data. This function contains 4 important boolean variables, and 1 boolean value is returned. In order to gain a deeper understanding of this function, I used var\_dump() to display the values of all variables. This helps me to examine if the function is working exactly as expected, so I can make any changes necessary. **In the following tests, variables are returned in this order:**

1. **\$timeValid**
2. **\$lunchValid**
3. **\$sessionValid**
4. **\$dayExamValid**
5. **\$meetingRegulations (returned to program)**

Purpose of test	Test data	Expected Result	Successful?
Testing with all valid data	time = 09:35 dayExamTime = 35 sessionTime = 35	(timeValid=) true (lunchValid=) true (sessionValid=) true	Yes

	numCompletedBreaks = 0	(dayExamValid=) true (meetingRegulations=) true	
--	------------------------	--	--

```
C:\wamp64\www\soundbro\schedule.php:47:boolean true
C:\wamp64\www\soundbro\schedule.php:47:boolean true
C:\wamp64\www\soundbro\schedule.php:47:boolean true
C:\wamp64\www\soundbro\schedule.php:47:boolean true
C:\wamp64\www\soundbro\schedule.php:83:boolean true
```

Purpose of test	Test data	Expected Result	Successful?
Testing with further valid data	time = 13:43 dayExamTime = 201 sessionTime = 66 numCompletedBreaks = 2	(timeValid=) true (lunchValid=) true (sessionValid=) true (dayExamValid=) true (meetingRegulations=) true	Yes

```
boolean true
boolean true
boolean true
boolean true
boolean true
```

Purpose of test	Test data	Expected Result	Successful?
Testing with valid boundary data (boundary for timeValid, sessionValid,	time = 17:00 dayExamTime = 390 sessionTime = 120 numCompletedBreaks = 3	(timeValid=) true (lunchValid=) true (sessionValid=) true (dayExamValid=) true (meetingRegulations=) true	Yes

and dayExamValid)			
----------------------	--	--	--

```
:boolean true
:boolean true
:boolean true
:boolean true
:boolean true
```

Purpose of test	Test data	Expected Result	Successful?
Testing with further valid boundary data (boundary for lunchValid)	time = 12:45 dayExamTime = 186 sessionTime = 113 numCompletedBreaks = 1	(timeValid=) true (lunchValid=) true (sessionValid=) true (dayExamValid=) true (meetingRegulations=) true	No

```
:boolean true
:boolean false
:boolean true
:boolean true
:boolean false
```

lunchValid was returned as false. It should be true, as this is on the valid side of the boundary. Initially, I thought that this error was received as objects of the DateTime class also have a date associated with them. However, this would've caused an error during the first test. Also, the time is formatted to only give the hours and minutes, so this is not the cause of the error. I decided to inspect more closely. The error must be in the following line:

```
// decide if a lunch break must be scheduled or not
$lunchValid = ($numCompletedBreaks != 1) OR ($time->format('H:i') <= $latestLunchTime->format('H:i'));
```

This variable is made up of 2 separate logical conditions. Using var\_dump() I outputted both conditions to closely examine their values. Using the same test data, the first condition should be false, and the second condition should be true.

```
:boolean false  
:boolean true
```

This screenshot shows that both conditions have returned the correct answer. Therefore, the combination of these conditions using 'or' is causing the mistake. This is due to errors involved in the order of precedence of logical operators. After some research, I found that '||' has a higher precedence than 'or', and works in the same way. I replaced this in my code, as shown below:

```
// decide if a lunch break must be scheduled or not  
$lunchValid = ($numCompletedBreaks != 1) || ($time->format('H:i') <= $latestLunchTime->format('H:i'));
```

I repeated the test with the updated code.

Purpose of test	Test data	Expected Result	Successful?
Testing with further valid boundary data (boundary for lunchValid)	time = 12:45 dayExamTime = 186 sessionTime = 113 numCompletedBreaks = 1	(timeValid=) true (lunchValid=) true (sessionValid=) true (dayExamValid=) true (meetingRegulations=) true	Yes

```
:boolean true  
:boolean true  
:boolean true  
:boolean true  
:boolean true
```

Purpose of test	Test data	Expected Result	Successful?
Testing with invalid boundary data for timeValid	time = 17:01 dayExamTime = 186 sessionTime = 113 numCompletedBreaks = 3	(timeValid=) false (lunchValid=) true (sessionValid=) true (dayExamValid=) true (meetingRegulations =) false	Yes

```
:boolean false
:boolean true
:boolean true
:boolean true
:boolean false
```

Purpose of test	Test data	Expected Result	Successful?
Testing with invalid data for timeValid	time = 17:35 dayExamTime = 186 sessionTime = 113 numCompletedBreaks = 3	(timeValid=) false (lunchValid=) true (sessionValid=) true (dayExamValid=) true (meetingRegulations =) false	Yes

```
:boolean false
:boolean true
:boolean true
:boolean true
:boolean false
```

Purpose of test	Test data	Expected Result	Successful?
Testing with invalid boundary data for lunchValid	time = 12:46 dayExamTime = 186 sessionTime = 113 numCompletedBreaks = 1	(timeValid=) true (lunchValid=) false (sessionValid=) true (dayExamValid=) true (meetingRegulations =) false	Yes

```
boolean true
boolean false
boolean true
boolean true
boolean false
```

Purpose of test	Test data	Expected Result	Successful?
Testing with invalid data for lunchValid	time = 13:09 dayExamTime = 186 sessionTime = 113 numCompletedBreaks = 1	(timeValid=) true (lunchValid=) false (sessionValid=) true (dayExamValid=) true (meetingRegulations =) false	Yes

```
:boolean true
:boolean false
:boolean true
:boolean true
:boolean false
```

Purpose of test	Test data	Expected Result	Successful?
Testing with invalid boundary data for sessionValid	time = 11:01 dayExamTime = 121 sessionTime = 121 numCompletedBreaks = 0	(timeValid=) true (lunchValid=) true (sessionValid=) false (dayExamValid=) true (meetingRegulations =) false	Yes

```
boolean true
boolean true
boolean false
boolean true
boolean false
```

Purpose of test	Test data	Expected Result	Successful?
Testing with invalid data for sessionValid	time = 11:01 dayExamTime = 121 sessionTime = 151 numCompletedBreaks = 0	(timeValid=) true (lunchValid=) true (sessionValid=) false (dayExamValid=) true (meetingRegulations =) false	Yes

```

boolean true
boolean true
boolean false
boolean true
boolean false

```

Purpose of test	Test data	Expected Result	Successful?
Testing with invalid boundary data for dayExamValid	time = 16:01 dayExamTime = 391 sessionTime = 102 numCompletedBreaks = 3	(timeValid=) true (lunchValid=) true (sessionValid=) true (dayExamValid=) false (meetingRegulations =) false	Yes

```

boolean true
boolean true
boolean true
boolean false
boolean false

```

Purpose of test	Test data	Expected Result	Successful?
-----------------	-----------	-----------------	-------------

Testing with invalid boundary data for dayExamValid	time = 16:01 dayExamTime = 422; sessionTime = 102 numCompletedBreaks = 3	(timeValid=) true (lunchValid=) true (sessionValid=) true (dayExamValid=) false (meetingRegulations =) false	Yes
---	---	--	-----

```
boolean true
boolean true
boolean true
boolean false
boolean false
```

### 3.5.3 Review of stage 5

#### *Stage 5 summary*

- Developed meetingRegulations function

There is no direct validation in this function. This is because the entire function is viewed as validating each variable to check the validity of the schedule. Therefore, this function itself **is** validation, confirming the current schedule is meeting TCL regulations.

#### *Testing carried out*

- Tested with valid, boundary, and invalid data for all 4 boolean variables
- Tested the full functionality of meetingRegulations

All tests were successful in the end.

#### *Success criteria met*

Success criteria S1 has not fully been met, however this function has taken a quite significant step towards meeting it. As this function decides if a schedule is meeting TCL regulations or not, it can be used to produce a schedule that meets these regulations. This function will also contribute to the success of S1-8.

### *Changes to the design*

There were no significant changes to the design in this section.

### *Review of the project so far*

The meetingRegulations function is a crucial part of the scheduling process, and its success is an important milestone in the development of this project. More functions still need to be developed so that this function can be implemented into the system. The directory structure is displayed below. It has not changed since the previous stage (this function is stored in schedule.php).

```
soundbro/
|--- index.php
|--- login.php
|--- register.php
|--- register-student.php
|--- register-supervisor.php
|--- dashboard.php
|--- schedule.php
|--- logout.php
|--- styles.css
|--- includes/
    |--- db-connect.php
    |--- navbar.php
    |--- footer.html
|--- logo/
    |--- full.png (main logo)
    |--- favicon3.png
```

## 3.6 addBreak function

### 3.6.1 Developing

When called, this function will run to add a break, or to end the day. This function will only run if the proposed exam to be scheduled is breaking TCL regulations, or if it is time for a break. Therefore, if this function is run and all breaks have been completed (`numCompletedBreaks = 3`), it must be the end of the day. It is important to understand that whenever TCL regulations are being broken (`meetingRegulations=False`), the solution is always to schedule a break. This is because all TCL regulations are centred around when a break is scheduled. Therefore, if regulations are broken, and all breaks have been scheduled, the day must end. This will cause `eod=True`. (`eod = end of day`)

The schedule function will try to stick to the ideal break times calculated by `nextBreak` by scheduling breaks within a certain interval of these times. The `addBreak` function will not alter the ‘Schedule’ entity in the database. Instead, it will increment the `$time` object by 15 or 60 minutes. This will cause the next exam to start 15/60 minutes after the previous exam ended. Breaks will only be recognised clearly when the schedule is displayed on screen. This is because the end time of the previous exam will not be equal to the start time of the next exam, which will lead to a break in the table to signify a break.

This function has 2 parameters: `time` and `numCompletedBreaks`. It will increment both of these accordingly, and return them. If a break is scheduled, the current session has ended and a new one is started. This will cause `sessionTime = 0` to be returned. Finally, the boolean variable `eod` is returned to signify it is the end of the day or not. If this variable is true, the while loop in the `schedule` function will stop, and a message will be displayed to say that the day has ended.

In a similar way to the validation in `nextBreak`, I decided to validate the input of `numCompletedBreaks`. If the input is invalid, it will return an error that will cause the while loop in the `schedule` function to end.

```
// validate input
if ($numCompletedBreaks >= 0 and $numCompletedBreaks <= 3) {
```

The `DateTime` class has a method ‘`modify`’ which allows the time to be incremented. I used this method to schedule the 15 / 60 minute break. As the algorithm for `addBreak` in section [2.5.2](#) shows, the value of `numCompletedBreaks` decides what the function will do. Each distinct value of this variable should result in a distinct action, so I have used a `switch/case` statement. This is a better structure with enhanced readability, as it clearly expresses the conditional logic associated with each value of `numCompletedBreaks`. This contributes to more modular and maintainable code.

```

switch($numCompletedBreaks){
    case 0:
        // 15 min break
        $time->modify('+15 minutes');

```

The full function is displayed below. As PHP does not allow multiple items to be returned, I must return them as an array.

```

function addBreak($time,$numCompletedBreaks){
    $eod=False;
    $sessionTime=0; // scheduling a break, so current session ends
    $error="";
    // validate input
    if ($numCompletedBreaks >= 0 and $numCompletedBreaks <= 3) {
        switch($numCompletedBreaks){
            case 0:
                // 15 min break
                $time->modify('+15 minutes');
                $numCompletedBreaks+=1;
                break;
            case 1:
                // 60 min lunch break
                $time->modify('+60 minutes');
                $numCompletedBreaks+=1;
                break;
            case 2:
                // 15 min break
                $time->modify('+15 minutes');
                $numCompletedBreaks+=1;
                break;
            case 3:
                // end of day
                $eod=True;
                break;
        }
    } else {
        // invalid numCompletedBreaks
        $error = "ERROR - Invalid number of completed breaks. ";
    }
    return [$time,$sessionTime,$numCompletedBreaks,$eod,$error]
}

```

### 3.6.2 Unit testing

All of the returned variables are displayed to the screen, allowing me to clearly see the outputs of the function. I used var\_dump() to do this. **In the following tests, variables are returned in this order:**

1. time
2. sessionTime
3. numCompletedBreaks

4. eod
5. error

```
var_dump($time->format('H:i'),$sessionTime,$numCompletedBreaks,$eod,$error);
```

\$time has been formatted to only output the hours and minutes, returning a string instead of the entire object. In the following tests, \$numCompletedBreaks is changed more as this variable determines which part of the switch/case statement will run. \$time is simply incremented, so cannot have any invalid or boundary data. It is always valid. Any validation involving the time is in other functions, such as meetingRegulations.

Purpose of test	Test data	Expected Result	Successful?
Testing with invalid (boundary) numCompletedBreaks	time = 10:10 numCompletedBreaks = 4	time = '10:10' sessionTime = 0 numCompletedBreaks = 4 eod = false error = 'ERROR - Invalid number of completed breaks.'	Yes

```
string '10:10' (Length=5)

int 0

int 4

boolean false

string 'ERROR - Invalid number of completed breaks.' (Length=44)
```

Purpose of test	Test data	Expected Result	Successful?
Testing with invalid numCompletedBreaks	time = 10:10 numCompletedBreaks = "two"	time = '10:10' sessionTime = 0 numCompletedBreaks= 'two' eod = false error = 'ERROR - Invalid number of completed breaks.'	Yes

```

string '10:10' (Length=5)

int 0

string 'two' (Length=3)

boolean false

string 'ERROR - Invalid number of completed breaks. '

```

Purpose of test	Test data	Expected Result	Successful?
Scheduling the first break (valid data)	time = 10:43 numCompletedBreaks = 0	time = '10:58' sessionTime = 0 numCompletedBreaks = 1 eod = false error = ''	Yes

```

:string '10:58' (Length=5)

:int 0

:int 1

:boolean false

:string '' (Length=0)

```

Purpose of test	Test data	Expected Result	Successful?
Scheduling the second break (valid data)	time = 12:31 numCompletedBreaks = 1	time = '13:31' sessionTime = 0 numCompletedBreaks = 2 eod = false error = ''	Yes

```

string '13:31' (Length=5)

int 0

int 2

boolean false

string '' (Length=0)

```

Purpose of test	Test data	Expected Result	Successful?
Scheduling the third break (valid data)	time = 15:10 numCompletedBreaks = 2	time = '15:25' sessionTime = 0 numCompletedBreaks = 3 eod = false error = ''	Yes

```

:string '15:25' (Length=5)

:int 0

:int 3

:boolean false

:string '' (Length=0)

```

Purpose of test	Test data	Expected Result	Successful?
Setting the end of day (valid data)	time = 16:57 numCompletedBreaks = 3	time = '16:57' sessionTime = 0 numCompletedBreaks = 3 eod = true error = ''	Yes

```

string '16:57' (Length=5)

int 0

int 3

boolean true

string '' (Length=0)

```

### 3.6.3 Review of stage 6

#### *Stage 6 summary*

- Developed the addBreak function to schedule breaks

#### *Testing carried out*

- Tested with valid and invalid data to examine all possible directions the algorithm can take

All tests were successful.

#### *Success criteria met*

The development of this function brings this solution very close to S1 being met. I cannot yet say that it has been met, as a schedule is not being created yet. However, this function ensures that the schedule will meet TCL regulations by scheduling breaks appropriately. This success criteria will be met once all of the functions specified in section 2.5.2 have been integrated into the schedule function.

S11 involves scheduling breaks automatically when editing the schedule. This is part of the edit function, which has been removed from the project. In future iterations of this solution, addBreak will be a useful function to allow S11 to be met. Additionally, S1-11 are all closer to being met due to the success of this stage.

#### *Changes to the design*

This function includes some extra validation of the variable numCompletedBreaks, which was not included in the design. This makes the system more robust, and allows any errors to be displayed clearly.

#### *Review of the project so far*

Creating a valid exam schedule cannot occur without this stage being successful. I am now able to schedule breaks correctly, and when combined with previous stages my solution should now contain most of the requirements needed to create a schedule that meets the

TCL requirements. The directory structure has not changed since the last stage, addBreak is stored in schedule.php along with nextBreak and meetingRegulations.

```
soundbro/
|--- index.php
|--- login.php
|--- register.php
|--- register-student.php
|--- register-supervisor.php
|--- dashboard.php
|--- schedule.php
|--- logout.php
|--- styles.css
|--- includes/
    |--- db-connect.php
    |--- navbar.php
    |--- footer.html
|--- logo/
    |--- full.png (main logo)
    |--- favicon3.png
```

## 3.7 accompAvailable function

### 3.7.1 Developing and testing

This function is run by schedule() to check if a certain accompanist is available at a certain time. It will return true if the accompanist is available, or false if not. It has 2 parameters: time and ExamineeID. It will also make some queries to the database to retrieve accompanist information. At this stage, the database has already been validated carefully. **Please note:** As with all stages, I am developing this using my algorithm from section 2.5. However, the **variable names used here are slightly different to those used in the flowchart.**

This stage is quite important, and quite difficult. Therefore, I will be conducting short tests throughout the development of this function before conducting a larger unit test. As per the flowchart in [2.5.2](#), the first step is to get the AccomplID and Duration of the given ExamineeID. Retrieving AccomplID is relatively simple, but retrieving duration was much trickier. Luckily, I had already created a query for this during stage 1 (section [3.1](#)). In that section I tested the database by conducting tests that are likely to be used in the solution. I used phpMyAdmin in that section to execute the query. I slightly modified it, and then used it in the function below. It uses several 'JOIN' commands to combine all the necessary tables. As the Timings table has a composite primary key, Timings.InstrumentFamily had to be linked to Examinees (through the Families table) **and** Timings.Grade had to be linked to Examinees.Grade.

```
function accompAvailable($ExamineeID,$time){  
  
    // get Duration + AccomplID of examinees exam  
    $examineeinfo="SELECT Duration,Examinees.AccompID  
    FROM timings  
    JOIN families ON families.InstrumentFamily = timings.InstrumentFamily  
    JOIN examinees ON examinees.Instrument = families.Instrument AND timings.Grade=examinees.Grade  
    WHERE ExamineeID='".$ExamineeID"';  
    $examineeResult=$conn->query($examineeinfo);  
    $row=$examineeResult->fetch_assoc();  
  
    //retrieve variables from SQL result  
    $duration=$row['Duration'];  
    $AccompID=$row['AccompID'];  
  
    var_dump($duration,$AccompID);  
}
```

From stage 1, I still have fake data stored in my database. I can use this for testing purposes in this section. I tried to output the variables for this function, given examineeID is 3. After checking the database, this examinee has a 23 minute exam with AccomplID 1. I expect duration to be 23 and AccomplID to be 1.



Warning: Undefined variable \$conn in C:\wamp64\www\soundbro\schedule.php on line 94

Call Stack

This resulted in this warning, saying that \$conn has not been recognised. \$conn is an object that creates the connection to the database. This later gave a fatal error, as shown below.

( ! )

Fatal error: Uncaught Error: Call to a member function query() on null in C:\wamp64\www\soundbro\schedule.php on line 94

As \$conn has not been recognised, the file db-connect.php must not be included correctly. This file stores php code that sets up a connection to the database using the MySQLi extension. It is a separate file to avoid repeating code across several pages. I suspected that the path referenced in the 'include' function was incorrect, leading to the file not connecting correctly.

```
include 'includes/db-connect.php';
```

Therefore, I tried copying and pasting the code within this file directly into schedule.php. This did not work. Then, I finally realised that this is a function, so I cannot access the db-connect.php code unless parameters are passed, or the 'include' is run inside this function. This is because variables are local in scope. I adjusted my code by making \$conn a third parameter for this function.

```
function accompAvailable($ExamineeID,$time,$conn){
```

Upon running this code again, var\_dump() outputted \$duration and \$accompID to the screen, as shown below.

```
:string '23' (Length=2)  
:string '1' (Length=1)
```

This matches my expected output, so the test is successful. The next process in the flowchart is to check if accompID is null. I used the if statement below to return true if this is the case, as an exam with no accompanist means infinite availability (ie. the 'accompanist' is no one, so the exam can happen at any time).

```
if (!$accompID){  
| return true;  
} else {
```

If accomID is not null, we need to find accompanist availability. To do this, the first step is to know which fields to check. Remember that availability is measured in 30 minute fields in the table ExamAccomps, and the names of these fields are Start0900, Start0930, etc. where the time in the field name is the start time of the 30 minute slot. Therefore, given a time, we need to round down to the nearest 30 minutes to find its appropriate slot. We need to do this for the exam's start time, end time, and midpoint. By making sure the accompanists are available at these 3 times, we know they are available during the entire exam. I created a separate function to find the time rounded down to the nearest 30 mins.

```
function down30mins($time){  
  
    // round down minutes to nearest 30  
    $minute = $time->format('i');  
    $roundedMinute = floor($minute / 30) * 30;  
  
    // create new object with rounded minutes  
    $roundDownTime = clone $time;  
    $roundDownTime->setTime(  
        $time->format('H'),  
        $roundedMinute,  
        $time->format('s')  
    );  
  
    return $roundDownTime;  
}
```

First, it is crucial to understand that if a time is rounded down, its hours don't change. 11:27 rounds down to 11:00, 14:43 rounds down to 14:30, 10:00 rounds down to 10:00. Therefore, we just need to round down the number of minutes. The floor function rounds down to the nearest integer, so `floor($minute / 30) * 30` is used. This divides the number of minutes by 30, so now it represents 'how many 30 minutes are in this many minutes'. This is a decimal number between 0 and 2 (2 not included). By rounding down to the nearest integer, it will either return 0 or 1, which is multiplied by 30 to give 0 or 30.

This roundedMinute is used to create a new time, roundDownTime, which is then returned back to accompAvailable. This function is run to calculate the start, middle and end times rounded to the nearest 30 minutes. I tested this by using `var_dump()` to output the inputs and outputs to the screen. Inputs are at the top, and outputs are at the bottom.

'10:43'

'10:30'

'11:12'

'11:00'

'15:00'

'15:00'

'16:59'

'16:30'

'13:30'

'13:30'

This function is working well, so now I can continue to format the times to get the corresponding field name. This rounded time, HH:MM, needs to go into the form 'StartHHMM'. I originally was going to format each time individually, but I have now decided to incorporate this into the function. The time rounded down would only be returned to create a field name for each time. This would cause 3 lines of code to be repeated, instead it is simpler and a better structure to return the appropriate field name in this function. Therefore, I will rename 'down30mins' to 'getAccompField'. This function will now take a time, and return an appropriate field name in ExamAccomps which relates to accompanist availability at that time.

The process of rounding down the time works well, this has been tested already. I just need to format this time appropriately to get the field name as 'StartHHMM' where H is hours and M is minutes. By concatenating the string 'Start' with the appropriately formatted time, I have done this.

```
// format time as 'StartHHMM'  
$formattedTime = 'Start' . $roundDownTime->format('Hi');  
  
return $formattedTime;
```

The method format('Hi') converts the object \$roundDownTime into a string. Within the brackets, the form of this string is defined. H represents the hours, and i represents the minutes, so HHMM is given. I tested this function, using a range of inputs.

'10:43' (Le

'Start1030'

'11:12' (Le

'Start1100'

'15:00' (Le

'Start1500'

'16:59' (Le

'Start1630'

'13:30' (Le

'Start1330'

'17:01' (Le

'Start1700'

Therefore, the function getAccompField is working well. It takes the time, and returns the corresponding field in ExamAccomps for availability. The full function is shown below.

```

function getAccompField($time){

    // round down minutes to nearest 30
    $minute = $time->format('i');
    $roundedMinute = floor($minute / 30) * 30;

    // create new object with rounded minutes
    $roundDownTime = clone $time;
    $roundDownTime->setTime(
        $time->format('H'),
        $roundedMinute,
        $time->format('s')
    );

    // format time as 'StartHHMM'
    $formattedTime = 'Start' . $roundDownTime->format('Hi');
    return $formattedTime;
}

```

In the function accompAvailable, if accompID is not null the following code will run:

```

// create object for middle and end time
$midTime = clone $time;
$halfDuration=$duration / 2;
$midTime->modify("+{$halfDuration} minutes");

$endTime = clone $time;
$endTime->modify("+{$duration} minutes");

//get field name for start/mid/end time
$startField=getAccompField($time); // $time is the start time of the exam
$midField=getAccompField($midTime);
$endField=getAccompField($endTime);

```

This part of the code will find the start, middle and end times of the exam, and find the corresponding field in the ExamAccomps table for each. However, I noticed one possible mistake in this process. The time 17:01 returns Start1700. The fields of availability only exist between Start0900 and Start1630, because exams cannot occur after 5pm (not even if they end at 17:01). Therefore, there needs to be some validation to check that startField, midField and endField are not 'Start1700'. If any of the fields are equal to this, then return false. There is no need to check any other fields such as 'Start1730'. This can be justified by considering the worst case scenario. The longest exam is 32 minutes. The latest time that will ever be inputted into accompAvailable is 5pm (any later and the while loop in schedule() will cause the process to end). For this case, startField is 'Start1700', midField is 'Start1700', and endField is 'Start1730'. The first 2 fields would cause the function to return false, so there is no point in checking anything else.

```
if ($startField=='Start1700' || $midField=='Start1700' || $endField=='Start1700'){
|    return false;
```

This additional validation prevents SQL queries from trying to get data from a field which does not exist. Now, we are sure that the fields are between Start0900 and Start1630, so we can retrieve data from ExamAccomps. It is likely that 2 or 3 of the fields are the same, but this does not affect how the function works.

```
} else {
// get availability from DB
$availabilitySQL="SELECT $startField,$midField,$endField
FROM ExamAccomps
WHERE AccompID = $accompID";
$availabilityResult=$conn->query($availabilitySQL);
$availabilityRow=$availabilityResult->fetch_assoc();
```

The SQL will retrieve the appropriate fields from ExamAccomps for the examinee's accompanist. \$availabilityResult stores the result of this query, and fetch\_assoc() creates an array of the result returned. In the database, boolean fields such as Start0900 - Start1630 are stored under the data type "tinyint(1)". This means that the field stores a 1, meaning true, or a 0, meaning false. Therefore, if all of the fields equal 1, the accompanist is available during the entire exam. In this case, the function returns true. If any fields equal 0, the accompanist is not available for the whole exam, so false is returned.

```
// store 3 variables to indicate availability at each point
$available_start = $availabilityRow[$startField];
$available_mid = $availabilityRow[$midField];
$available_end = $availabilityRow[$endField];

if ($available_start==1 and $available_mid==1 and $available_end==1){
    // if accompanist is available the during the whole exam
    return true;
} else {
    // if accompanist is not available during exam at any point
    return false;
}
```

The full function, accompAvailable, is shown below.

```

function accompAvailable($ExamineeID,$time,$conn){

    // get Duration + AccompID of examinees exam
    $examineeinfo="SELECT Duration,Examinees.AccompID
    FROM timings
    JOIN families ON families.InstrumentFamily = timings.InstrumentFamily
    JOIN examinees ON examinees.Instrument = families.Instrument AND timings.Grade=examinees.Grade
    WHERE ExamineeID=$ExamineeID";
    $examineeResult=$conn->query($examineeinfo);
    $row=$examineeResult->fetch_assoc();

    // retrieve variables from SQL result
    $duration=$row['Duration'];
    $accompID=$row['AccompID'];

    if (!$accompID){
        // examinee has no accompanist
        return true;
    } else {

        // create object for middle time
        $midTime = clone $time;
        $halfDuration=$duration / 2;
        $midTime->modify("+{$halfDuration} minutes");

        // create object for end time
        $endTime = clone $time;
        $endTime->modify("+{$duration} minutes");

        //get field name for start/mid/end time
        $startField=getAccompField($time); // $time is the start time of the exam
        $midField=getAccompField($midTime);
        $endField=getAccompField($endTime);
    }
}

```

```

if ($startField=='Start1700' || $midField=='Start1700' || $endField=='Start1700'){
    // exams run past 5pm - this breaks regulations so return false
    return false;
} else {
    // get availability from ExamAccomps
    $availabilitySQL="SELECT $startField,$midField,$endField
    FROM ExamAccomps
    WHERE AccompID = $accompID";
    $availabilityResult=$conn->query($availabilitySQL);
    $availabilityRow=$availabilityResult->fetch_assoc();

    // store 3 variables to indicate availability at each point
    $available_start = $availabilityRow[$startField];
    $available_mid = $availabilityRow[$midField];
    $available_end = $availabilityRow[$endField];

    if ($available_start==1 and $available_mid==1 and $available_end==1){
        // if accompanist is available the during the whole exam
        return true;
    } else {
        // if accompanist is not available during exam at any point
        return false;
    }
}
}

```

### 3.7.2 Unit testing

In stage 1 of development, I populated the database with fake data. This data will be used for unit testing in this stage. There are 3 if statements in this function, each one nested in another. I will test valid, invalid and boundary data (where possible) for this function. This function now has 3 parameters: ExamineeID, time, and conn. conn stores the connection to the database, this input will not be changed throughout all testing. However, the other variables will be. By using different combinations of examinees and their accompanists at different times, I can systematically test the full functionality of this unit. As the database is also an input, some relevant records and fields will be displayed in ‘test data’. An inputted ExamineeID will always exist in the database, as the accompAvailable function is run when reading from a list of ExamineeIDs taken from the database. I thought it would be unnecessary to validate this, when there is no possibility of invalid data. The databases have been validated already, so an AccomplID in Examinees will always exist in ExamAccomps, and so on. This includes all validation specified in [2.4.1 - examinees](#).

Using var\_dump(), I will display the values of available\_start, available\_mid, and available\_end. Not all tests will lead to these values being calculated, as some scenarios do not need availability to return an answer. I will also display the returned value of the function. This allows me to get a better understanding of how the function is working, so any correct outputs with incorrect logic can be detected, before they create larger errors in the full system later on.

If outputted, the 3 variables are displayed in the order: available\_start, available\_mid, and available\_end. After this, the returned value is displayed.

Purpose of test	Test data	Expected Result	Successful?
Testing the function when the accompanist is available (with valid data)	<p>(\$conn is constant through all tests)</p> <p>\$ExamineeID=1</p> <ul style="list-style-type: none"><li>• AccomplID = 3</li><li>• Start0900 = 1</li></ul> <p>\$time=09:00</p>	<p>available_start=1 available_mid=1 available_end=1 accompAvailable=true</p>	Yes

```
:string '1' (
:string '1' (
:string '1' (
:boolean true
```

Purpose of test	Test data	Expected Result	Successful?
When accompanist is available at another time (valid boundary)	\$ExamineeID=1 • AccomplID = 3 • Start1130 = 1  \$time=11:37	available_start=1 available_mid=1 available_end=1 accompAvailable=true	Yes

```
:string '1' (
:string '1' (
:string '1' (
:boolean true
```

Purpose of test	Test data	Expected Result	Successful?
Testing the function when the accompanist is available across 2 time slots (with valid data)	\$ExamineeID=1 • AccomplID = 3 • Start0930 = 1 • Start1000 = 1  \$time=09:54	available_start=1 available_mid=1 available_end=1 accompAvailable=true	Yes

```
string '1' (
string '1' (
string '1' (
:boolean true
```

Purpose of test	Test data	Expected Result	Successful?
Testing the function when the accompanist is available across 3 time slots (with valid data)  This is the maximum possible	\$time=14:59  \$ExamineeID=8 • Duration = 32 • AccomplID = 2 • Start1430 = 1 • Start1500 = 1 • Start1530 = 1	available_start=1 available_mid=1 available_end=1 accompAvailable=true	Yes

number of different time slots.			
---------------------------------	--	--	--

```
string '1' (
string '1' (
string '1' (
boolean true
```

Purpose of test	Test data	Expected Result	Successful?
Testing the function when examinee has no accompanist	\$time=12:00 \$ExamineeID=2 <ul style="list-style-type: none"> <li>Duration = 13</li> <li>AccompID = null</li> </ul>	accompAvailable=true	Yes

:boolean true

Purpose of test	Test data	Expected Result	Successful?
Testing the function when an exam runs past 5pm	\$time=16:43 \$ExamineeID=14 <ul style="list-style-type: none"> <li>Duration = 23</li> <li>AccompID = 3</li> <li>Start1630 = 1</li> </ul>	accompAvailable=false	Yes

:boolean false

Purpose of test	Test data	Expected Result	Successful?
Testing the function when an exam with no accompanist	\$time=16:55 \$ExamineeID=2 <ul style="list-style-type: none"> <li>Duration = 13</li> </ul>	accompAvailable=false	No

runs past 5pm	• AccomID = null		
---------------	------------------	--	--

:boolean true
---------------

When this test failed, I had a lot of trouble deciding whether or not to fix it. This function, accompAvailable, will decide whether or not an accompanist is available at a certain time. I modified my design to prevent Start1700 from being retrieved (which does not exist). However, I coincidentally implemented validation to prevent exams being scheduled past 5pm. This is not necessary though. After accompAvailable returns true, meetingRegulations would catch if an exam runs past 5pm. It would then schedule the end of the day, or try to squeeze in one more exam. Therefore, if an examinee with no accompanist is going past 5pm (like in this test above), one would think there is not much need to return false now, as the mistake will be caught later.

However, I chose to fix this error for a number of reasons. Firstly, catching errors earlier prevents larger logical errors which can cause severe detriment to the solution.

Secondly, it is crucial that as many exams are scheduled as possible. If accompAvailable detects some exams that are breaking regulations, the for loop in schedule() will simply go to the next examinee in ExamineeIDs. However, if meetingRegulations detects an exam is breaking regulations, it will schedule a break or run the squeeze function. There is a subtle but important difference between these 2 situations:

- If accompAvailable detects the 5pm error:
  - It checks the next examinee in ExamineeIDs
  - This list is sorted by priority, so if the next examinee is scheduled, it is the most optimal for the efficiency of the schedule. If this examinee is scheduled, there is a higher chance of scheduling all exams.
- If meetingRegulations detects the 5pm error:
  - It may run the squeeze function
  - This function will try to schedule an exam that goes as close to 5pm as possible, in order to waste as little time as possible. This is predominantly a good thing. However, it does not consider the priority of exams.

For the most part, this difference is very minimal. However, I plan to develop a system that schedules exams across multiple days in the future. If the squeeze function does not incorporate the priority of exams in its logic, it may schedule an exam over one with a higher priority. The next day, this unscheduled exam's accompanist may not be available, and that exam could be left unscheduled. Therefore, I have chosen to take remedial action so that the above test will return false.

Now I am clear on my intentions: if an exam runs past 5pm, return false **no matter what**. Currently, this decision is nested within the if statement to separate examinees with and without accompanists. I need to reverse this order and the problem should be solved.

```
if ($startField=='Start1700' || $midField=='Start1700' || $endField=='Start1700'){
    // exams run past 5pm - this breaks regulations so return false
    return false;
} else {
    if (!$accompID){
        // examinee has no accompanist
        return true;
    } else {
```

This modified code should fix the issue. I ran the test again to check.

Purpose of test	Test data	Expected Result	Successful?
Testing the function when an exam with no accompanist runs past 5pm	\$time=16:55 \$ExamineeID=2 <ul style="list-style-type: none"><li>• Duration = 13</li><li>• AccomplID = null</li></ul>	accompAvailable=false	Yes

!:**boolean false**

Purpose of test	Test data	Expected Result	Successful?
Testing the function when an accompanist is completely unavailable	\$time=15:22 \$ExamineeID=3 <ul style="list-style-type: none"><li>• Duration = 23</li><li>• AccomplID = 1</li><li>• Start1500 = 0</li><li>• Start1530 = 0</li></ul>	available_start=0 available_mid=0 available_end=0 accompAvailable=false	Yes

```
:string '0' (L
:string '0' (L
:string '0' (L
:boolean false
```

Purpose of test	Test data	Expected Result	Successful?
Testing the function when an exam ends at 5pm (boundary data)	\$time=16:42 \$ExamineeID=7 • Duration = 18 • AccomplID = 4 • Start1630 = 1  (endtime is 5pm)	available_start=1 available_mid=1 available_end=1 accompAvailable=true	No

```
boolean false
```

If an exam ends at 5pm exactly, it is allowed. This is the only case where endField is Start1700 and true is returned. Therefore, I changed the code to account for this case.

```
if (($startField=='Start1700' || $midField=='Start1700' || $endField=='Start1700')
    and ($endTime->format('H:i')!='17:00')) {
    // exams run past 5pm - this breaks regulations so return false
    return false;
} else {
```

The if statement will not execute if the exam ends at 5pm. However, endField still stores 'Start1700', which does not exist. To prevent errors, there is code later on to fix this.

```
if ($endTime->format('H:i')=='17:00'){
    // if exam ends at 5pm, remove 'Start1700'
    $endField=$midField;
}
// get availability from ExamAccomps
```

By setting endField to midField, there should be no errors, and all fields stored in start/mid/endField should be between Start0900 and Start1630 (they should all be **valid**). I repeated the test to check this.

Purpose of test	Test data	Expected Result	Successful?
Testing the function when an exam ends at 5pm (boundary data)	\$time=16:42 \$ExamineeID=7 • Duration = 18 • AccomplID = 4 • Start1630 = 1  (endtime is 5pm)	available_start=1 available_mid=1 available_end=1 accompAvailable=true	Yes

```
:string '1' (
:string '1' (
:string '1' (
:boolean true
```

Purpose of test	Test data	Expected Result	Successful?
Testing the function when an accompanist is partially unavailable	\$time=10:22 \$ExamineeID=10 • Duration = 18 • AccomplID = 1 • Start1000 = 0 • Start1030 = 1  (endtime is 5pm)	available_start=0 available_mid=1 available_end=1 accompAvailable=false	Yes

```
:string '0' (L
:string '1' (L
:string '1' (L
:boolean false
```

### 3.7.3 Review of stage 7

#### *Stage 7 summary*

- Created accompAvailable function
- Also, created getAccompField function which is called by accompAvailable

#### *Testing carried out*

- Tested with valid, invalid, and boundary data
- Tested with varying times (including exams ending before, during and after 5pm)
- Tested with varying availability (partially and fully available / unavailable)

All tests were successful in the end.

#### *Success criteria met*

This stage has allowed the program to decide if an accompanist is available at a given time. This allows the schedule to adhere to accompanist availability, so S6 has been met.

This function also helps to detect clashes when editing the schedule, which brings me closer to meeting S10. Although the edit function will not be built during this project, accompAvailable will form a modular part of the edit function in future versions of the system.

S1-11 are closer to being met, as this function will be incorporated into the schedule() function and the edit() function. Some changes made during unit testing should lead to the schedule() function grouping exams more, and scheduling even more exams. This means that S3 and S6 are especially close to being met.

#### *Changes to the design*

I have made several changes to the design in this section. Variable names have changed, as the function has become more complex. I have included an extra parameter (\$conn) to allow the function to communicate with the database. The process of rounding down times and formatting fields have been combined into the new function 'getAccompField'. This function is more efficient than the original design, and better structured. Without it, some code would be repeated 3 times each, making it more difficult to maintain and less readable.

I have included some validation in this function that was originally not included. If the exam runs past 5pm, it returns false. This will happen whether or not the examinee has an accompanist. This validation would occur later in the process, but by doing it earlier there is a better chance of more exams being scheduled. This reasoning is explained in more detail during my unit testing for this section.

## *Review of the project so far*

The solution should now contain all complexity that relates to the TCL requirements. However, these functions are not yet integrated into the schedule function. The most important function, recalculateOrder, will be developed next. This will develop the logic associated with grouping exams **and** sorting exams by priority so that the schedule is as efficient as possible. Very soon, I should be able to combine all these functions into one by developing the schedule function. I will develop the squeeze function after the schedule function, and integrate it into the prototype afterwards. This is so that I can test how the schedule is functioning at a lower level before including the extra variable of the squeeze function. This should give clarity as to how the prototype is working at a more fundamental scale, allowing me to fix any errors before they become too large. The directory structure has not changed since the previous stage and this function is also stored in schedule.php.

```
soundbro/
|--- index.php
|--- login.php
|--- register.php
|--- register-student.php
|--- register-supervisor.php
|--- dashboard.php
|--- schedule.php
|--- logout.php
|--- styles.css
|--- includes/
    |--- db-connect.php
    |--- navbar.php
    |--- footer.html
|--- logo/
    |--- full.png (main logo)
    |--- favicon3.png
```

## 3.8 recalculateOrder function

### 3.8.1 Developing and testing

This function is very important to the success of this solution. It contains the most complexity of all designed algorithms, and should be the key to scheduling exams efficiently. This function should sort an array of ExamineeIDs by priority and also group them appropriately. The priority of an examinee's exam depends entirely on their accompanist's current "free time". Their free time can be calculated as available time - exam time. As examinees are scheduled, and time increments, each accompanist's free time (and therefore priority) will change. This function calculates free time for each accompanist, and therefore it needs to be run whenever an exam / break is scheduled. This is why it is run at the start of the large while loop in the schedule function. Once free time has been calculated, the examinees are sorted by their accompanist's freetime (ascending). This automatically groups examinees by accompanist. Within each accompanist group, examinees are grouped by instrument family. Within each instrument family, examinees are grouped by instrument. Within each instrument, examinees are sorted ascending by grade. This will all be done by the function recalculateOrder, which will return the sorted list back to schedule(), as well as an error message if necessary. By sorting exams by priority, more urgent exams can be scheduled first, increasing the chance of all exams being scheduled. By grouping exams in this way, the schedule should roughly be grouped in that order, which would meet Ms Pearcey's requirements.

I am developing this as designed in the flowchart in section 2.5.2. The image below shows the start of this function. As I will be using databases in this function, \$conn is also passed as a parameter, as it defines the connection to the database using the MySQLi extension.

```
function recalculateOrder($examineeIDs,$time,$conn){  
    $accompIDs=[];  
    $error=" ";  
  
    for ($i = 0; $i < count($examineeIDs); $i++) {  
        $ExamineeID=$examineeIDs[$i];  
        // get AccomID and Duration for examinee
```

#### Calculating exam time

The first part of this function needs to calculate the total exam time for each accompanist. Within the for loop, I need to get the AccomID and Duration of a given examinee. This exact process was required when I developed the accompAvailable function in the previous stage. To repeat this code would make the program less efficient, and less readable. It would also be harder to maintain, as I would have to make the same change across all occurrences of this code. Therefore, I will develop a new function, called "getExamineeInfo". This will be used by accompAvailable and recalculateOrder, improving the structure of the program and reducing repetition. This makes my solution more modular.

```

function getExamineeInfo($ExamineeID,$conn){

    // get Duration + AccompID of examinees exam
    $examineeinfo="SELECT Duration,Examinees.AccompID
    FROM timings
    JOIN families ON families.InstrumentFamily = timings.InstrumentFamily
    JOIN examinees ON examinees.Instrument = families.Instrument AND timings.Grade=examinees.Grade
    WHERE ExamineeID=$ExamineeID";
    $examineeResult=$conn->query($examineeinfo);
    $row=$examineeResult->fetch_assoc();

    // retrieve variables from SQL result
    $duration=$row['Duration'];
    $accompID=$row['AccompID'];

    return [$duration,$accompID];
}

```

I copied the relevant code from accompAvailable to create this function. By removing the code and calling this function in accompAvailable, I tested that the outputs were still as expected. These tests were successful, giving me confidence that the function is working well.

```

for ($i = 0; $i < count($examineeIDs); $i++) {
    $ExamineeID=$examineeIDs[$i];
    // get AccompID and Duration for examinee
    [$duration,$accompID]=getExamineeInfo($ExamineeID,$conn);

```

I called this function in recalculateOrder to obtain the accompID and duration. Using the following code, I tested that this function was working correctly by displaying the duration and accompID on the screen.

```

if (!$accompID){
    $accompID="NULL";
}
echo $duration.<br>.$accompID,<br><br>;

```

16
3
13
NULL
23
1
13
5

These numbers were outputted for ExamineeIDs 1 to 4. This output matched the values I found on the database using phpMyAdmin, so this short test was successful, allowing me to move on. The flowchart for this function next specifies adding the accompID to the array accompIDs. However, it does not mention what to do if accompID is null (ie. the examinee has no accompanist). In the program, I will include code to ignore any null values of accompID. At the end of this function, all examinees with no accompanist will be appropriately appended to the array. As they have no accompanist, they have the most free time and therefore least urgency to be scheduled, so they are at the end of the array.

In this initial for loop, I process all examinees in examineeIDs. After retrieving accompID and duration, I check if accompID is null.

```
// if accompID null, skip this bit
if ($accompID){
```

If it is, then we go to the next examinee in the loop. If not, the array accompIDs is updated as follows:

- If the current accompID exists in the array, find its index. Increment accompIDs[index][1] by the duration.

```

$accompIndex = null;
// iterate through each accompanist
foreach ($accompIDs as $rIndex => $row) {
    // Check if this row matches the accompID
    if ($row[0] === $accompID) {
        // increment by duration
        $accompIndex = $rIndex;
        $accompIDs[$accompIndex][1] += $duration;
        break;
    }
}

```

(This is slightly more complex as it is a 2D array)

- If the current accompID is not found (accompIndex is null), add it to the array, and increment the duration in the same way

```

// If accompID not yet in accompIDs
if ($accompIndex === null) {
    // add to array, and increment by duration
    $accompIDs[] = [$accompID, 0, 0, 0];
    $accompIndex = count($accompIDs) - 1;
    $accompIDs[$accompIndex][1] += $duration;
}

```

We just appended something to accompIDs, so its index must count(accompIDs)-1.

The 2 other zeros will be used by this function. Currently, they are placeholders. This process is repeated for all examinees in examineeIDs. By the end of the for loop, accompIDs should store the total **remaining** exam time for each accompanist. This is remaining because the array ExamineeIDs only contains unscheduled examinees. If an examinee is scheduled, they are removed from this array and their accompanist's exam time will later decrease. Originally, this section was designed to search for accompID in accompIDs and add a new entry to the array if not found. Then, it would search through the array again to find the index of the accompID, and then increment by duration. As this carries out 2 searches, it is inefficient and unnecessarily complex. By searching for the index first, we are searching accompID anyway, which is a simpler and easier process.

This function is quite complex, so I will be running tests at some points during the development of this function. The first for loop is complete, so I will run a short test now. These low-level tests allow me to find and correct any errors earlier on in the development process. As I keep developing this function, I get closer to the final iteration of my function. With each iteration, I will be sure that it is working correctly so I have the confidence to trust that part of the function. The inputs to this function are \$examineeIDs, \$time, and \$conn. \$conn is a constant, and \$time is not used yet. \$examineeIDs can be changed to assess how the function works. Please note that this function will never be run if \$examineeIDs is

empty, and will always contain valid examineeIDs. Therefore, the different tests should alter the length of the array, and the number of null accompanists.

The test data will not include time and conn, as they are irrelevant to the output. For each test, I have calculated the expected output manually by viewing the data in phpMyAdmin. The array accompIDs is outputted to the screen.

Purpose of test	Test data	Expected Result	Successful?
Testing with 15 examinees in examineeIDs	(Using fake data in the database from stage 1)  examineeIDs = [1,2,3,4,5,6,7,8,9,10,11,12,13,14,15]	accompIDs= 3,41,0,0 1,41,0,0 5,26,0,0 2,78,0,0 4,31,0,0	Yes

```
accompIDs =
3, 41, 0, 0
1, 41, 0, 0
5, 26, 0, 0
2, 78, 0, 0
4, 31, 0, 0
```

Purpose of test	Test data	Expected Result	Successful?
Testing with 5 examinees in examineeIDs	(Using fake data in the database from stage 1)  examineeIDs = [1,2,3,4,5]	accompIDs= 3,16,0,0 1,23,0,0 5,13,0,0 2,23,0,0	Yes

```
accompIDs =
3, 16, 0, 0
1, 23, 0, 0
5, 13, 0, 0
2, 23, 0, 0
```

Purpose of test	Test data	Expected Result	Successful?
Testing where all examinees have no accompanist	(Using fake data in the database from stage 1) examineeIDs = [2,6,13,15]	accomplIDs= (null)	Yes

accomplIDs =

```

function recalculateOrder($examineeIDs,$time,$conn){
    $accompIDs=[];
    $error="";

    //calculate total exam time
    for ($i = 0; $i < count($examineeIDs); $i++) {

        $ExamineeID=$examineeIDs[$i];
        // get AccompID and Duration for examinee
        [$duration,$accompID]=getExamineeInfo($ExamineeID,$conn);

        // if accompID null, do not add to accompIDs
        if ($accompID){

            $accompIndex = null;
            // iterate through each accompanist
            foreach ($accompIDs as $rIndex => $row) {
                // Check if this row matches the accompID
                if ($row[0] === $accompID) {
                    // increment by duration
                    $accompIndex = $rIndex;
                    $accompIDs[$accompIndex][1] += $duration;
                    break;
                }
            }
            // If accompID not yet in accompIDs
            if ($accompIndex === null) {
                // add to array, and increment by duration
                $accompIDs=[[$accompID,0,0,0]];
                $accompIndex=count($accompIDs)-1;
                $accompIDs[$accompIndex][1] += $duration;
            }
        }
    }
}

```

Here is the function `recalculateOrder` so far. The current iteration of this function has passed all tests for this point. I trust that it correctly calculates the total remaining exam time for each accompanist.

**Currently, accompIDs contains [AccompID,examTime,0,0].**

### *Calculating available time*

The next for loop will loop through every row in accompIDs, in order to find the availableTime and freeTime for each accompanist. For each accompanist, the first set of instructions are concerned with obtaining an associative array of availability for the accompanist. To calculate the available time, only the availability after the current time should be considered.

Using the MySQLi connection, I made an associative array called \$availability for the current accompanist.

```
// get associative array of availability  
$availability=$availabilityResult->fetch_assoc();
```

The next step is to check if the field corresponding to the current time is true. E.g if the time is 11:13, check Start1100. If the time is 13:51, check Start1330. This process has already been done before, and coincidentally I have already made it into a separate function! This function is called 'getAccompField(\$time)'. Using this function, I was able to return the '\$currentField'. However, I also need to be able to calculate the difference between the current time and the next time slot. In order to do this, I need the time that has been rounded down inside the getAccompField function. Therefore, I have modified this function so that it returns \$roundDownTime as well as \$currentField. I have also modified all occurrences where this function is called (I ran accompAvailable a few times after making these changes, to ensure that nothing was affected. The tests were all successful.)

```
// get rounded time, and field corresponding to time  
[$roundDownTime,$currentField]=getAccompField($time);  
//adjust roundDownTime so it stores time of next field  
$roundDownTime->modify('+30 minutes');  
  
//find difference between current time and next 30min slot  
$difference = $time->diff($roundDownTime);  
$difference=$difference->i; //format so it stores number of minutes  
  
for ($i = 0; $i < count($accompIDs); $i++){
```

Calling getAccompField simplifies the code and improves modularity in the solution. The currentField and roundDownTime depend only on the time. They do not depend on the value of accompID, so they have been placed outside of the for loop. If the current accompanist is available at the current time, then I must calculate how many minutes are left in the current time slot, so that this number can be added to their available time. In order to do this, I increase roundDownTime by 30 minutes so that it stores the time of the next 30 min slot. By using diff() to find the difference between \$time and \$roundDownTime, I can find the number of minutes until the next slot starts. This is stored in \$difference. Then, the for loop of accompIDs can start by creating the associative array of accompanist availability.

To test that \$difference was storing the correct value, I outputted \$time and \$difference for a range of different times.

:string '10:43'	:string '15:59'	:string '16:00'	:string '13:38'
:int 17	:int 1	:int 30	:int 22

As these above screenshots show, the variable \$difference clearly stores the number of minutes left in the current time slot. This process is working successfully. \$difference will only be added to an accompanist's available time if they are available at the current time. As \$availability is an associative array of availability, and \$currentField represents the field corresponding to the current time, the code below will decide if an accompanist is available, and act accordingly. Each accompanist's available time is stored in accompIDs[i][2].

```
// if accompanist is currently available, add $difference to their available time
if ($availability[$currentField]==1){
    $accompIDs[$i][2]+=$difference;
}
```

All fields after the currentField need to be processed. If an accompanist is available for one of these fields, simply add 30 minutes to that accompanist's available time. My design specifies to remove fields before and including the currentField from \$availability. To do this, I will use array\_slice. This will create a new array that starts from a specified index. I need this index to be the index of currentField+1.

To find the index of currentField, I used array\_keys to get an array of all keys in \$availability, and then used array\_search to get the index of currentField in that array. Using this variable, I shortened the array using array\_slice so that it only contained fields that should be considered when incrementing the available time.

```
// shorten array to everything after currentField
$currentIndex = array_search($currentField, array_keys($availability));
$availability=array_slice($availability,$currentIndex+1);

foreach ($availability as $key => $value) {
    if ($availability[$key]==1){
        $accompIDs[$i][2]+=30;
    }
}
```

This loop will search through all relevant fields, and add 30 minutes to the available time whenever an accompanist is available during that time slot. This should end up with accompIDs[i][2] storing all the time available for each accompanist. I ran some more tests to confirm this before moving on. By reading the values of everyone's availability on the phpMyAdmin interface, I manually calculated the expected results. To test available time, time must be changed. The contents of examineIDs does **not** affect the available time of

each accompanist, so it has not been changed. It is constant through all tests as  
`examineeIDs = [1,2,3,4,5,6,7,8,9,10,11,12,13,14,15]` .

Purpose of test	Test data	Expected Result	Successful?
Testing that availableTime is correctly stored using a valid time (valid data)	(Using fake data in the database from stage 1)  time=13:38	accomplIDs= 3,41,0,0 1,41,82,0 5,26,52,0 2,78,150,0 4,31,82,0	Yes

```
accomplIDs =
3,41,0,0
1,41,82,0
5,26,52,0
2,78,150,0
4,31,82,0
```

Purpose of test	Test data	Expected Result	Successful?
Testing that availableTime is correctly stored when time is at the start of a time slot (boundary data)	(Using fake data in the database from stage 1)  time=10:00	accomplIDs= 3,41,180,0 1,41,270,0 5,26,270,0 2,78,270,0 4,31,300,0	Yes

```

accomplIDs =
3,41,180,0
1,41,270,0
5,26,270,0
2,78,270,0
4,31,300,0

```

Purpose of test	Test data	Expected Result	Successful?
Testing that availableTime is correctly stored when time is the latest possible time (boundary data)	(Using fake data in the database from stage 1)  time=17:00	accomplIDs= 3,41,0,0 1,41,0,0 5,26,0,0 2,78,0,0 4,31,0,0	No

**( ! )** Warning: Undefined array key "Start1700" in C:\wamp64\www\soundbro\schedule.php on line 237

Call Stack

There is no validation to check if the time is 17:00. All times from 09:00 to 16:59 should work fine, as currentField exists in the database. If the time is later than 17:00, this function will not be run. However, at 17:00 it can be run. This has caused getAccompField to return a currentField = 'Start1700'. When the array of database results are searched, the algorithm has tried to look for a field called Start1700 and returned this warning. If the time is 17:00, available time is always 0 for every accompanist, because exams must end at 5pm. I could develop some additional functionality to effectively skip most of this function if the time is 17:00. However, as I plan to develop this function to schedule exams over multiple days, this solution would be more of a 'quick fix'. In later versions of this system, the time 17:00 would be seen in a similar way to a break, simply causing the time to be incremented appropriately. Therefore, recalculateOrder should not stop carrying out its functionality if it is 17:00. It should just skip over the process of calculating the available time, as this is the cause of the error.

In my design, the current for loop should also calculate the free time. However, I will end the for loop here, and nest it inside an if statement. This should only calculate available time if the time is not 17:00. If it is, then it will not calculate available time for each accompanist (leaving it as 0), and then continue with the rest of the function.

```

if ($time->format('H:i')!="17:00"){
    //calculate available time
    for ($i = 0; $i < count($accompIDs); $i++){

```

I ran this test again to check if this has fixed the issue.

Purpose of test	Test data	Expected Result	Successful?
Testing that availableTime is correctly stored when time is the latest possible time (boundary data)	(Using fake data in the database from stage 1) time=17:00	accompIDs= 3,41,0,0 1,41,0,0 5,26,0,0 2,78,0,0 4,31,0,0	Yes

```

accompIDs =
3,41,0,0
1,41,0,0
5,26,0,0
2,78,0,0
4,31,0,0

```

```

//find difference between current time and next 30min slot
$difference = $time->diff($roundDownTime);
$difference=$difference->i; //format so it stores number of minutes

if ($time->format('H:i')!="17:00"){
    //calculate available time
    for ($i = 0; $i < count($accompIDs); $i++){
        // get availability for accompanist
        $accompID=$accompIDs[$i][0];
        $availabilitySQL="SELECT Start0900,Start0930,Start1000,Start1030,Start1100,
        Start1130,Start1200,Start1230,Start1300,Start1330,Start1400,
        Start1430,Start1500,Start1530,Start1600,Start1630
        FROM ExamAccomps
        WHERE AccompID=$accompID";

        $availabilityResult=$conn->query($availabilitySQL);

        // get associative array of availability
        $availability=$availabilityResult->fetch_assoc();

        // if accomp is currently available, add $difference to their available time
        if ($availability[$currentField]==1){
            $accompIDs[$i][2]+=$difference;
        }

        // shorten array to everything after currentField
        $currentIndex = array_search($currentField,array_keys($availability));
        $availability=array_slice($availability,$currentIndex+1);

        foreach ($availability as $key => $value) {
            //increment available time by 30 for every available slot
            if ($availability[$key]==1){
                $accompIDs[$i][2]+=30;
            }
        }
    }
}

```

All updates to the function since the last iteration have been displayed. This code displays the process of calculating the available time for each accompanist. This test is successful, as are all of the tests, so I can move on to the next step: calculating free time. I can now trust that this prototype of my function is reliable and robust, which should reduce the chance of errors later on.

**Currently, accompIDs contains [AccompID,examTime,availableTime,0].**

## Calculating free time

As I have explained in my design, the free time is a great indicator of priority as it takes into account everyone's availability without being too sensitive to it. It is also good for finding accompanists with insufficient availability. If an accompanist has to spend more time playing exams then they are available for, this function should return an error as the accompanist either has too many exams, or not enough availability. In this situation, free time would be negative. Therefore, checking if free time is positive or negative is a great measure of which accompanists need to adjust their availability. After designing my scheduling algorithms in section [2.5.2](#), Ms Pearcey mentioned that she liked this feature, saying "I like the check to ensure the accompanist has enough time available". Therefore, I hope to develop this correctly to reduce her workload and make this project more successful.

In my design, calculating free time occurs in the same for loop that calculates the available time. However, creating validation if time is 17:00 has caused this loop to be nested inside an if statement. If it is 17:00, free time should still be calculated, so I must create an additional for loop outside of the if statement.

```
//calculate free time
for ($i = 0; $i < count($accompIDs); $i++){
    $accompID=$accompIDs[$i][0];
    $examTime = $accompIDs[$i][1];
    $availableTime = $accompIDs[$i][2];
    $freeTime = $availableTime - $examTime;
```

This code calculates freeTime clearly as availableTime - examTime. If freeTime is negative, I need to formulate my error message. This code will run for all accompanists, so the message must be appended to the string \$error. This will allow Ms Pearcey to see all accompanists with insufficient availability, not just the first one. If free time is negative, then they need at least freetime \* -1 minutes more availability to solve the issue.

```
if ($freeTime < 0){
    $minutes_more=$freeTime*-1;
```

The first and last name of the relevant accompanist is then retrieved from the database. Using these variables, I add to \$error with the message below.

```
//update error message with relevant info
$error.="$firstName $lastName has too many exams / is not available enough.
They need at least $minutes_more minutes more. <br>";
```

This message is clear and direct, and provides the most useful information to help Ms Pearcey take action. My design specifies that free time should be left 0 in the array if it is negative. However, I find it more beneficial to store free time exactly as calculated, even if

that includes negative numbers. This means that the list will precisely reflect the most urgent exams, making my algorithm more accurate.

```
}
```

```
// update array to include freeTime
```

```
$accompIDs[$i][3]=$freeTime;
```

The 2D array accompIDs should now store free time in the 3rd index of each row. To confirm that this process has been coded correctly, I ran some tests. By displaying \$error and \$accompIDs, I can assess how this iteration of the function is working.

Purpose of test	Test data	Expected Result	Successful?
Testing that freeTime and error are correctly displayed when one person has freeTime < 0.	(Using fake data in the database from stage 1)  examineeIDs = [1,2,3,4,5,6,7,8,9,10,11,12 ,13,14,15]  time=13:38	Christopher Smith has too many exams / is not available enough. They need at least 41 minutes more.  accompIDs= 3,41,0,-41 1,41,82,41 5,26,52,26 2,78,150,72 4,31,82,51	Yes

Christopher Smith has too many exams / is not available enough. They need at least 41 minutes more.

```
accompIDs =
```

```
3,41,0,-41
```

```
1,41,82,41
```

```
5,26,52,26
```

```
2,78,150,72
```

```
4,31,82,51
```

Purpose of test	Test data	Expected Result	Successful?
Testing that freeTime and error are correctly displayed when many people have freeTime < 0	(Using fake data in the database from stage 1)  examineeIDs = [1,2,3,4,5,6,7,8,9,10,11,12 ,13,14,15]  time=15:37	Christopher Smith has too many exams / is not available enough. They need at least 41 minutes more.  David Williams has too many exams / is not available enough. They need at least 41 minutes more.	Yes

		Brian Davis has too many exams / is not available enough. They need at least 26 minutes more.  accomplIDs= 3,41,0,-41 1,41,0,-41 5,26,0,-26 2,78,83,5 4,31,60,29	
--	--	---	--

Christopher Smith has too many exams / is not available enough. They need at least 41 minutes more.  
 David Williams has too many exams / is not available enough. They need at least 41 minutes more.  
 Brian Davis has too many exams / is not available enough. They need at least 26 minutes more.

accomplIDs =  
3,41,0,-41  
1,41,0,-41  
5,26,0,-26  
2,78,83,5  
4,31,60,29

Purpose of test	Test data	Expected Result	Successful?
Testing that freeTime and error are correctly displayed when one person has freeTime = 0	(Using fake data in the database from stage 1)  examineeIDs = [1,2,3,4,5,6,7,8,9,10,11,12 ,13,14,15]  time=12:19	accomplIDs = 3,41,41,0 1,41,161,120 5,26,131,105 2,78,150,72 4,31,161,130	Yes

accomplIDs =  
3,41,41,0  
1,41,161,120  
5,26,131,105  
2,78,150,72  
4,31,161,130

```

//calculate free time
for ($i = 0; $i < count($accompIDs); $i++){
    $accompID=$accompIDs[$i][0];
    $examTime = $accompIDs[$i][1];
    $availableTime = $accompIDs[$i][2];
    $freeTime = $availableTime - $examTime;

    if ($freeTime < 0){
        $minutes_more=$freeTime*-1;

        //get name from DB
        $nameSQL="SELECT FirstName, LastName
        FROM Supervisors
        WHERE SupervisorID = $accompID";
        $nameResult=$conn->query($nameSQL);
        $nameRow=$nameResult->fetch_assoc();
        $firstName=$nameRow['FirstName'];
        $lastName=$nameRow['LastName'];

        //update error message with relevant info
        $error.="$firstName $lastName has too many exams / is not available enough.
        They need at least $minutes_more minutes more. <br>";
    }
    // update array to include freeTime
    $accompIDs[$i][3]=$freeTime;
}

}

```

This section of recalculateOrder displays the newly developed process of calculating free time. The tests above show that this function is working fine, it is able to correctly calculate free time for each accompanist and store an error message whenever needed.

**Currently, accompIDs contains [AccompID,examTime,availableTime,freeTime].**

### *Sorting by free time*

Now that I have a list of accompIDs and their free times, I can sort ascending by freeTime. This allows me to sort them by priority in order to schedule exams more efficiently. To sort this array, I used the usort() function. This function sorts an array using a user-defined comparison function. Therefore, I had to define a separate function, which I named 'freeTimeSort'. This takes 2 parameters, rowA and rowB. These 2 parameters are consecutive rows in the array. Noting that rowA[3] / rowB[3] is freeTime, if rowA[3] is less than rowB[3] ( $\text{rowA}[3] - \text{rowB}[3] < 0$ ), then rowA should come before rowB. If rowA[3] is greater than rowB[3] ( $\text{rowA}[3] - \text{rowB}[3] > 0$ ), rowA should go after rowB. The usort function will handle this processing, but my function 'freeTimeSort' needs to return the difference between rowA[3] and rowB[3], in order to give it a positive or negative number. This number will allow usort to sort accompIDs ascending by freeTime.

```
function freeTimeSort($rowA, $rowB) {
    return $rowA[3] - $rowB[3];
}
```

```
// sort ascending by free time
usort($accompIDs, 'freeTimeSort');
```

This should now sort arrays ascending by free time. I outputted the array before and after running this function to test if it worked correctly.

BEFORE

```
accompIDs =  
3,41,41,0  
1,41,161,120  
5,26,131,105  
2,78,150,72  
4,31,161,130
```

AFTER

```
accompIDs =  
3,41,41,0  
2,78,150,72  
5,26,131,105  
1,41,161,120  
4,31,161,130
```

This shows that the 3rd index of each row is ascending, so it has been sorted by free time correctly. Now, my function correctly sorts accompanists by priority! This prototype version of this function is working accurately, but there is a final feature left to develop. The next step is to convert this into a sorted array of examineeIDs which is grouped by the instrument family, then instrument, then grade.

#### *Sorting by family, instrument and grade*

All examinees should be sorted by their accompanist's free time, which groups them into accompanist blocks automatically. This is so that more important examinees are higher up in the list, so can be scheduled quicker. The additional sorting must occur within each accompanist block. This is why I have created a new for loop that iterates through each row in accompIDs. In order to sort by these factors, I need to create an array that stores the ExamineeID, InstrumentFamily, Instrument, and Grade for all examinees in examineeIDs

with the current accompanist. To create this array, I first need to find all examinees in examineeIDs who have the current accompanist (accompIDs[i]). Therefore, I have created a second for loop that loops through examineeIDs inside of the first loop.

```
$newExamineeIDs=[];
//convert sorted accompIDs into sorted examineeIDs
for ($i = 0; $i < count($accompIDs); $i++){
    $temp_array=[];

    for ($i=0; $i < count($examineeIDs); $i++){

        //find examinees in examineeIDs with current accompanist
```

In this loop, I select the AccomplID of the current examinee and check if it is the same as the current accompanist's ID. If it is, then I make a complex SQL query to retrieve the ExamineeID, InstrumentFamily, Instrument, and Grade for this examinee. The result is converted into an associative array, which is added to temp\_array. Once the for loop of examinees is complete, temp\_array should store all the relevant information needed to do the relevant sorting.

```
for ($i=0; $i < count($examineeIDs); $i++){

    //find examinees in examineeIDs with current accompanist
    $accompSQL="SELECT AccomplID FROM Examinees
    WHERE ExamineeID=$examineeIDs[$i]";
    $accompResult=$conn->query($accompSQL);
    $accompRow=$accompResult->fetch_assoc();
    $examinee_accomp=$accompRow['AccomplID'];

    if ($examinee_accomp==$accompIDs[$i][0]){
        //add their info to temp_array
        $instrumentSQL="SELECT Examinees.ExamineeID,
        Families.InstrumentFamily,Families.Instrument,Timings.Grade
        FROM timings
        JOIN families ON families.InstrumentFamily = timings.InstrumentFamily
        JOIN examinees ON examinees.Instrument=families.Instrument
        AND timings.Grade=examinees.Grade
        WHERE examinees.ExamineeID=$examineeIDs[$i]";
        $instrumentResult=$conn->query($instrumentSQL);
        $temp_array[]=$instrumentResult->fetch_assoc();
    }
}
```

I tested that temp\_array stored the correct values and had the correct structure by running the function and displaying it to the screen using var\_dump(). Unfortunately, temp\_array was empty and it displayed the following warnings.

( ! ) Warning: Undefined array key 3 in C:\wamp64\www\soundbro\schedule.php on line 323

( ! ) Warning: Trying to access array offset on value of type null in C:\wamp64\www\soundbro\schedule.php on line 323

These warnings point to an issue on line 323.

323			if (\$examinee_accomp==\$accompIDs[\$i][0]) {
-----	--	--	---

I thought that accompIDs may not be stored correctly as a 2D array, but previous prototypes have used \$accompIDs[\$i][0] and returned no errors. Therefore, the issue must be with the variable \$i. During this entire process, I had forgotten that this for loop is running inside another for loop. Therefore, I should be using 2 different variables. Currently, the variable \$i is being overwritten by the second loop, which is resulting in accompIDs trying to access indexes larger than their range. I simply changed the relevant variables to \$j in the second for loop, as shown below. I was careful to only make this change for variables linked to examineeIDs, not accompIDs, to prevent a similar error from occurring.

```
for ($j=0; $j < count($examineeIDs); $j++){  
  
    //find examinees in examineeIDs with current accompanist  
    $accompSQL="SELECT AccompID FROM Examinees  
    WHERE ExamineeID=$examineeIDs[$j];  
    $accompResult=$conn->query($accompSQL);  
    $accompRow=$accompResult->fetch_assoc();  
    $examinee_accomp=$accompRow['AccompID'];  
  
    if ($examinee_accomp==$accompIDs[$i][0]){  
        //add their info to temp_array  
        $instrumentSQL="SELECT Examinees.ExamineeID,  
        Families.InstrumentFamily,Families.Instrument,Timings.Grade  
        FROM timings  
        JOIN families ON families.InstrumentFamily = timings.InstrumentFamily  
        JOIN examinees ON examinees.Instrument=families.Instrument  
        AND timings.Grade=examinees.Grade  
        WHERE examinees.ExamineeID=$examineeIDs[$j];  
        $instrumentResult=$conn->query($instrumentSQL);  
        $temp_array[]=$instrumentResult->fetch_assoc();  
    }  
}
```

The highlighted variables indicate where the changes were made. This distinguishes the indexes for accompIDs and examineeIDs, which should fix the error. I ran the same data again to check. This successfully created a temp\_array for each accompanist, part of this 2D array is shown below.

```

array (size=3)
0 =>
array (size=4)
'ExamineeID' => string '1' (Length=1)
'InstrumentFamily' => string 'piano' (Length=5)
'Instrument' => string 'piano' (Length=5)
'Grade' => string '4' (Length=1)
1 =>
array (size=4)
'ExamineeID' => string '3' (Length=1)
'InstrumentFamily' => string 'guitar' (Length=6)
'Instrument' => string 'classical guitar' (Length=16)
'Grade' => string '6' (Length=1)
2 =>
array (size=4)
'ExamineeID' => string '7' (Length=1)
'InstrumentFamily' => string 'woodwind' (Length=8)
'Instrument' => string 'saxophone' (length=9)
'Grade' => string '5' (Length=1)

```

temp\_array now must be sorted by InstrumentFamily, then Instrument, then grade. As this is quite a specific sort, I have again used the function usort(). I must therefore create an additional function which takes 2 elements in temp\_array, and returns a number. If 0, the order should not change. If positive, the first element should go after the second element and if negative, the second should go after the first. This function will be called 'instrumentSort'.

```
//sort by family > instrument > grade
usort($temp_array, 'instrumentSort');
```

```
function instrumentSort($a, $b) {
```

For simplicity, the 2 parameters passed to instrumentSort are called \$a and \$b. These represent 2 elements in the array. The key idea here is to realise that instrument family and instrument do not necessarily have to be 'sorted', but just have to be grouped together. Therefore, the way in which the algorithm sorts different values is irrelevant. After some research I found the function strcmp(), which compares each character in the 2 strings based on their ASCII values. If the strings are equal, strcmp returns 0. If the first string is less than the second string, it returns a negative value, and vice versa. Therefore, if 2 instrument families are the same, strcmp returns 0.

```
// sort by instrument family
$result = strcmp($a['InstrumentFamily'], $b['InstrumentFamily']);
```

If the result is 0, I need to compare the different instruments of the 2 elements. The same logic used for the instrument family applies here. If they are the same instrument, strcmp() returns 0. If not, it returns a positive/negative value that will be used to help usort() to sort the array.

```
// if family is the same, sort by instrument
if ($result === 0) {
| $result = strcmp($a['Instrument'], $b['Instrument']);
}
```

Finally, if strcmp() returns 0 again, then the instruments are the same. In this case, the elements are sorted ascending by grade. If the first element has a higher grade than the second, they should be swapped (so instrumentSort should return a positive value), and vice versa. Therefore, returning the first element's grade minus the second element's grade is enough to sort ascending by grade.

```
// if instrument is the same, sort ascending by grade
if ($result === 0) {
| $result = $a['Grade'] - $b['Grade'];
}

return $result;
```

This function now contains all the necessary logic to correctly sort any 2 elements in temp\_array. Using the usort() function, the entire array temp\_array can be sorted. Once sorted, temp\_array is appended to newExamineeIDs and the loop repeats for the next accompanist.

Once the for loop has added examinees for all accompanists, the list is grouped by accompanist and sorted by priority. Within each accompanist block, it is grouped into families, then instruments, and finally each instrument is sorted by grade. The final process of this function is to finally include a group that has been neglected since the start: examinees with no accompanist.

Examinees without an accompanist do not have to worry about availability, so they are always at the lowest priority to be scheduled. Therefore, they are added at the end of the newExamineeIDs. However, they must still be sorted using instrumentSort and usort.

To sort examinees with no accompanist, most of the code has already been developed. temp\_array is reset, and the for loop through examineeIDs is started. Looping through all examinees in examineeIDs, the function tries to find examinees where AccomplID = null. If so, their information is added to temp\_array. Once the loop is complete, temp\_array is sorted

using usort and finally appended to newExamineeIDs. newExamineeIDs now contains all examinees that were inputted, in the correct order, as specified in the design section.

```
0 =>
array (size=4)
  'ExamineeID' => string '12' (Length=2)
  'InstrumentFamily' => string 'brass' (Length=5)
  'Instrument' => string 'trombone' (Length=8)
  'Grade' => string '2' (Length=1)

1 =>
array (size=4)
  'ExamineeID' => string '5' (Length=1)
  'InstrumentFamily' => string 'brass' (Length=5)
  'Instrument' => string 'trumpet' (Length=7)
  'Grade' => string '7' (Length=1)
```

The array newExamineeIDs currently looks like the above screenshot. I need to extract the ExamineeID only. To do this, I will create one final array: finalExamineeIDs. This will store the IDs only. For each sub-array in newExamineeIDs, finalExamineeIDs will append subarray['ExamineeID'].

```
//extract just the ExamineeIDs
$finalExamineeIDs = [];
foreach ($newExamineeIDs as $subArray) {
    $finalExamineeIDs[] = $subArray['ExamineeID'];
}

return [$finalExamineeIDs,$error];
}
```

This, as well as the error, is returned to the schedule function. This function should now be complete. To test this function I ran several tests using different combinations of data, including valid, invalid, and boundary. Currently my database is populated with fake data. I adjusted some of this data throughout these tests to appropriately examine the full range of each function. This includes changes to instruments, accompanists and availability. It would be unnecessarily complicated to list all changes made for each test, but the expected output column is always accurate, and has been manually calculated by viewing the data stored in the databases using phpMyAdmin. I have outputted examineeIDs using echo, and included the code below to only display the error if it contains a message.

```

<!-- display if error is found -->
<?php if (isset($error) and $error!="") { ?>
|   <div class="alert alert-danger">
|     <?php echo $error; ?>
|   </div>
<?php } ?>

```

As previous iterations of this function have been tested, these tests will focus more on the recent developments to the code.

Purpose of test	Test data	Expected Result	Successful?
Testing general functionality (all accompanists available)	(Using modified fake data in database) time=10:33 examineeIDs=[1,2,3,4,5,6,7,8,9,10,11,12,13,14,15]	12,7,13,9,8,2,11,6,4,15,1,3,10,5,14	Yes

[12 , 7 , 13 , 9 , 8 , 2 , 11 , 6 , 4 , 15 , 1 , 3 , 10 , 5 , 14]

This function takes a list of integers, and outputs that list of integers in a different order. As a result, it can be difficult to prove that this function actually works. Therefore, for the first test, I have manipulated the function to give a slightly more informative result. This should clearly show that the sorting process was effective.

### accompIDs

```

C:\wamp64\www\soundbro\schedule.php:427:
array (size=2)
0 =>
array (size=4)
  0 => string '2' (Length=1)
  1 => int 126
  2 => int 237
  3 => int 111
1 =>
array (size=4)
  0 => string '1' (Length=1)
  1 => int 98
  2 => int 240
  3 => int 142

```

There are 2 accompanists for these examinees. Remember that, in each smaller array, [0] is the AccomplID, and [3] is the free time. The accompanist with AccomplID=2 has less free

time than AccomID=1. Therefore, all examinees where AccomID=2 are put at the start of the list.

## examineeIDs

```
C:\wamp64\www\soundbro\schedule.php:429:  
array (size=15)  
0 =>  
array (size=5)  
'ExamineeID' => string '12' (Length=2)  
'AccompID' => string '2' (Length=1)  
'InstrumentFamily' => string 'brass' (length=5)  
'Instrument' => string 'trombone' (length=8)  
'Grade' => string '2' (Length=1)  
1 =>  
array (size=5)  
'ExamineeID' => string '7' (length=1)  
'AccompID' => string '2' (Length=1)  
'InstrumentFamily' => string 'brass' (length=5)  
'Instrument' => string 'trombone' (length=8)  
'Grade' => string '5' (Length=1)  
2 =>  
array (size=5)  
'ExamineeID' => string '13' (Length=2)  
'AccompID' => string '2' (Length=1)  
'InstrumentFamily' => string 'brass' (length=5)  
'Instrument' => string 'trombone' (length=8)  
'Grade' => string '5' (Length=1)  
3 =>  
array (size=5)  
'ExamineeID' => string '9' (Length=1)  
'AccompID' => string '2' (Length=1)  
'InstrumentFamily' => string 'brass' (length=5)  
'Instrument' => string 'trumpet' (length=7)  
'Grade' => string '1' (length=1)
```

```

4 =>
  array (size=5)
    'ExamineeID' => string '8' (length=1)
    'AccompID' => string '2' (length=1)
    'InstrumentFamily' => string 'brass' (length=5)
    'Instrument' => string 'trumpet' (length=7)
    'Grade' => string '8' (length=1)

5 =>
  array (size=5)
    'ExamineeID' => string '2' (length=1)
    'AccompID' => string '2' (length=1)
    'InstrumentFamily' => string 'strings' (length=7)
    'Instrument' => string 'violin' (length=6)
    'Grade' => string '2' (length=1)

6 =>
  array (size=5)
    'ExamineeID' => string '11' (length=2)
    'AccompID' => string '2' (length=1)
    'InstrumentFamily' => string 'strings' (length=7)
    'Instrument' => string 'violin' (length=6)
    'Grade' => string '6' (length=1)

7 =>
  array (size=5)
    'ExamineeID' => string '6' (length=1)
    'AccompID' => string '1' (length=1)
    'InstrumentFamily' => string 'brass' (length=5)
    'Instrument' => string 'french horn' (length=11)
    'Grade' => string '1' (length=1)

```

This is the ordered array of examineeIDs, but containing lots of extra information for each examinee. As this shows, the accompanist with less free time is placed earlier (AccompID 2). Within this block for AccompID 2, all 5 brass exams are grouped, followed by 2 strings exams. The 3 trombone exams and 2 trumpet exams are grouped within the brass section. Trombone exams are sorted ascending by grade (grade 2 then 5 then 8), and so are trumpets (grade 1 then 8). The 2 strings exams are both violins, so they are also ascending by grade (grade 2 then 6). The order of instrument blocks and instrument family blocks is irrelevant within their respective groups, as mentioned earlier.

```

8 =>
array (size=5)
  'ExamineeID' => string '4' (length=1)
  'AccompID' => string '1' (length=1)
  'InstrumentFamily' => string 'guitar' (length=6)
  'Instrument' => string 'classical guitar' (length=16)
  'Grade' => string '3' (length=1)

9 =>
array (size=5)
  'ExamineeID' => string '15' (length=2)
  'AccompID' => string '1' (length=1)
  'InstrumentFamily' => string 'guitar' (length=6)
  'Instrument' => string 'classical guitar' (length=16)
  'Grade' => string '3' (length=1)

10 =>
array (size=5)
  'ExamineeID' => string '1' (length=1)
  'AccompID' => string '1' (length=1)
  'InstrumentFamily' => string 'guitar' (length=6)
  'Instrument' => string 'classical guitar' (length=16)
  'Grade' => string '4' (length=1)

11 =>
array (size=5)
  'ExamineeID' => string '3' (length=1)
  'AccompID' => string '1' (length=1)
  'InstrumentFamily' => string 'guitar' (length=6)
  'Instrument' => string 'classical guitar' (length=16)
  'Grade' => string '6' (length=1)

```

```

12 =>
array (size=5)
  'ExamineeID' => string '10' (length=2)
  'AccompID' => string '1' (length=1)
  'InstrumentFamily' => string 'strings' (length=7)
  'Instrument' => string 'violin' (length=6)
  'Grade' => string '4' (length=1)

13 =>
array (size=5)
  'ExamineeID' => string '5' (length=1)
  'AccompID' => null
  'InstrumentFamily' => string 'brass' (length=5)
  'Instrument' => string 'trumpet' (length=7)
  'Grade' => string '7' (length=1)

14 =>
array (size=5)
  'ExamineeID' => string '14' (length=2)
  'AccompID' => null
  'InstrumentFamily' => string 'guitar' (length=6)
  'Instrument' => string 'classical guitar' (length=16)
  'Grade' => string '7' (length=1)

```

From index 7 to 12 is the block of exams for AccomplID = 1. A single french horn exam is placed first, followed by 4 guitar exams and 1 violin exam. The 4 guitars are all classical guitar exams, so they have been sorted by grade (grade 3 then 3 then 4 then 6). The french horn and violin exist alone in their instrument family, so cannot be grouped any further. Finally, we have 2 exams with no accompanist. Correctly, they have been placed last (as they have the least priority). These 2 exams have different families, so cannot be grouped any more than currently shown. However, the next test will show that sorting is still accurate for examinees with no accompanist. This test distinctly shows that the order of examineeIDs is sorted exactly as designed, from sorting by accompanists all the way down to sorting by grade.

Purpose of test	Test data	Expected Result	Successful?
Testing when all examinees have no accompanist	(Using modified fake data in database)  time=13:00 examineeIDs=[1,2,3,4,5,6,7,8]	5,8,6,4,1,3,2,7	Yes
[5 , 8 , 6 , 4 , 1 , 3 , 2 , 7]			

Purpose of test	Test data	Expected Result	Successful?
Testing when all examinees have an accompanist	(Using modified fake data in database)  time=09:00 examineeIDs=[9,10,11,12,13,14,15]	12,13,15,10,9,11,14	Yes
[12 , 13 , 15 , 10 , 9 , 11 , 14]			

Purpose of test	Test data	Expected Result	Successful?
Testing when examineeIDs is just one	(Using modified fake data in database)	1	Yes

examinee (boundary data)	time=13:00 examineeIDs=[1]		
-----------------------------	-------------------------------	--	--

[1]

This function will not conduct any validation to test if the array is empty, because this check will be done before running the function. It would be redundant to include this now.

Purpose of test	Test data	Expected Result	Successful?
Testing when examineeIDs input is non-sequential	(Using modified fake data in database)  time=10:50 examineeIDs=[1,8,4,2,12,10,5,3]	12,10,5,8,4,1,3,2	Yes

[12 , 10 , 5 , 8 , 4 , 1 , 3 , 2]

Purpose of test	Test data	Expected Result	Successful?
Testing functionality when some errors should be returned	(Using modified fake data in database)  time=14:33 examineeIDs=[1,2,3,4,5,6,7,8,9,10,11,12,13,14,15]	4,1,3,14,12,5,10,9,8,11,7,13,6,15,2  Brian Davis has too many exams / is not available enough. They need at least 77 minutes more.  David Williams has too many exams / is not available enough. They need at least 54 minutes more.	Yes (see below)

[4 , 1 , 3 , 14 , 12 , 5 , 10 , 9 , 8 , 11 , 7 , 13 , 6 , 15 , 2]

Brian Davis has too many exams / is not available enough. They need at least 77 minutes more.  
David Williams has too many exams / is not available enough. They need at least 54 minutes more.

This test has passed, and the expected output matches the actual output. However, I do not believe that it puts enough emphasis on a specific part of the error message. It is quite likely,

based on this scheduling algorithm, that 77 minutes will not be enough for Brian Davis, and 54 minutes will not be enough for David Williams. This many minutes of additional availability would cause free time = 0, so whenever they are available, their exam must be scheduled. If a break is scheduled, or another exam is scheduled over them, another error would be returned, only with less minutes needed. For example, after making the necessary changes, an error message might say ‘Brian Davis has too many exams / is not available enough. They need at least 40 minutes more.’ This message would be quite frustrating for Ms Pearcey to see, and would force her to go back to the accompanist and ask again, delaying the whole scheduling process and increasing her workload. Therefore, I will try to mitigate this issue by emphasising the words ‘at least’. More specifically, this will be emphasised by underlining it.

```
//update error message with relevant info
$error.="{$firstName $lastName has too many exams / is not available enough.
They need <u>at least</u> $minutes_more minutes more. <br>";
```

The <u> tag should make the message more clear to prevent misunderstandings. I ran this test again, using the same inputs.

Purpose of test	Test data	Expected Result	Successful?
Testing functionality when some errors should be returned (invalid data)	(Using modified fake data in database)  time=14:33 examineeIDs=[1,2,3,4,5,6,7,8,9,10,11,12,13,14,15]	4,1,3,14,12,5,10,9,8,11,7,1 3,6,15,2  Brian Davis has too many exams / is not available enough. They need <u>at least</u> 77 minutes more.  David Williams has too many exams / is not available enough. They need <u>at least</u> 54 minutes more.	Yes

[4 , 1 , 3 , 14 , 12 , 5 , 10 , 9 , 8 , 11 , 7 , 13 , 6 , 15 , 2]

Brian Davis has too many exams / is not available enough. They need at least 77 minutes more.  
David Williams has too many exams / is not available enough. They need at least 54 minutes more.

I am happy that this message now communicates the error message more effectively. It emphasises that the number of minutes is a bare minimum, without distracting from the overall message.

This testing has evaluated the success of the final iteration of recalculateOrder. As a result of this constant process of testing and improving, I am confident that this function is working correctly. Repeated unit testing throughout this section with varying inputs has tested an exhaustive range of possible outputs. This means that, when this function is integrated into the full solution, there is a smaller chance of errors. All developments to this function since the last test have been displayed below.

```
// sort ascending by free time
usort($accompIDs, 'freeTimeSort');

$newExamineeIDs=[];
//convert sorted accompIDs into sorted examineeIDs
for ($i = 0; $i < count($accompIDs); $i++){
    $temp_array=[];

    for ($j=0; $j < count($examineeIDs); $j++){

        //find examinees in examineeIDs with current accompanist
        $accompSQL="SELECT AccompID FROM Examinees
        WHERE ExamineeID=$examineeIDs[$j]";
        $accompResult=$conn->query($accompSQL);
        $accompRow=$accompResult->fetch_assoc();
        $examinee_accomp=$accompRow['AccompID'];

        if ($examinee_accomp==$accompIDs[$i][0]){
            //add their info to temp_array
            $instrumentSQL="SELECT Examinees.ExamineeID,
            Families.InstrumentFamily,Families.Instrument, Timings.Grade
            FROM timings
            JOIN families ON families.InstrumentFamily = timings.InstrumentFamily
            JOIN examinees ON examinees.Instrument=families.Instrument
            AND timings.Grade=examinees.Grade
            WHERE examinees.ExamineeID=$examineeIDs[$j]";
            $instrumentResult=$conn->query($instrumentSQL);
            $temp_array[]=$instrumentResult->fetch_assoc();
        }
    }

    //sort by family > instrument > grade
    usort($temp_array, 'instrumentSort');

    //temp_array is added to the end of newExamineeIDs
    $newExamineeIDs = array_merge($newExamineeIDs, $temp_array);
    //re-index the array so it is formatted correctly
    $newExamineeIDs = array_values($newExamineeIDs);
}
```

```

//repeat for examinees with no accompanist
$temp_array=[];
for ($i=0; $i < count($examineeIDs); $i++){

    //find examinees in examineeIDs with no accompanist
    $accompSQL="SELECT AccompID FROM Examinees
    WHERE ExamineeID=$examineeIDs[$i]";
    $accompResult=$conn->query($accompSQL);
    $accompRow=$accompResult->fetch_assoc();
    $examinee_accomp=$accompRow[ 'AccompID'];

    if ($examinee_accomp==null){
        //add their info to temp_array
        $instrumentSQL="SELECT Examinees.ExamineeID,Families.InstrumentFamily,
        Families.Instrument,Timings.Grade
        FROM timings
        JOIN families ON families.InstrumentFamily = timings.InstrumentFamily
        JOIN examinees ON examinees.Instrument = families.Instrument
        AND timings.Grade=examinees.Grade
        WHERE examinees.ExamineeID=$examineeIDs[$i]";
        $instrumentResult=$conn->query($instrumentSQL);
        $temp_array[]=$instrumentResult->fetch_assoc();
    }
}

//sort by family > instrument > grade
usort($temp_array, 'instrumentSort');

//temp_array is added to the end of newExamineeIDs
$newExamineeIDs = array_merge($newExamineeIDs, $temp_array);
//re-index the array so it is formatted correctly
$newExamineeIDs = array_values($newExamineeIDs);

//extract just the ExamineeIDs
$finalExamineeIDs = [];
foreach ($newExamineeIDs as $subArray) {
    $finalExamineeIDs[] = $subArray[ 'ExamineeID'];
}

return [$finalExamineeIDs,$error];

```

### 3.8.2 Review of stage 8

#### *Stage 8 summary*

- Developed `recalculateOrder` function
  - It calculates the total remaining exam time for each accompanist
  - It calculates the remaining time available for each accompanist
  - It calculates the free time for each accompanist
  - Using this, it sorts examinees by their accompanist's free time
  - It sorts examinees by instrument family → instrument → grade (ascending) within each accompanist block
  - It returns an error message for any accompanists who do not have enough time available
- Developed `getExamineeInfo` function for improved modularity
- Developed the functions `freeTimeSort` and `instrumentSort` in order to use the `usort()` function correctly

#### *Testing carried out*

Due to the size of this function, unit testing at the end of this stage did not seem appropriate. Instead I chose to conduct unit tests on each prototype of the function as it was developed, allowing me to correct errors more efficiently .

- Tested that it calculates exam time correctly
- Tested that it calculates available time correctly
- Tested that it calculates free time correctly
- Tested that it returns errors correctly (and later edited the error message for improved clarity)
- Tested against boundary data (eg. the latest possible time, 17:00, or shortest number of elements in examineeIDs, 1, or when free time = 0)
- Tested that the function sorts examineeIDs correctly by their accompanist's free time
- Tested that the function correctly sorts the entire array based on all criteria

I tested with a range of valid, invalid and boundary data. I also changed different values in the database to examine the full limits of the functionality. All tests were successful in the end.

#### *Success criteria met*

This function has completed several success criteria. S2 has been met, as the process of sorting by free time ensures that the most urgent examinees are scheduled first, so time is used as effectively as possible. S3 has almost been met. Using similar logic, one could argue that as many exams are scheduled as possible. This function will certainly contribute to scheduling as many exams as possible, but the squeeze function will ensure that S3 has been absolutely met.

S4 has been met, as the process of sorting examinees by their accompanist's free time automatically groups accompanists together. When examinees are scheduled, the schedule function attempts to maintain the order provided by this function (as it also considers

accompanist availability). This function contains all of the complexity involved in grouping accompanists in this way.

S5 is also complete, as the order of examinees within each block is sorted by family, then instrument, then grade (within each accompanist block). The function instrumentSort, combined with usort(), defines the method used to do this. As a result, the returned array of examineeIDs is sorted in this way.

Finally, success criteria S7 has been achieved, as the variable \$error contains an error message detailing all accompanists with insufficient availability. When calculating free time, all accompanists with free time < 0 are included in this error message. This is because they have to play in more exams than they have the time for, so more availability (or less exams) must be inputted.

Alternatively, the structure diagram clearly shows that section 2.1.1 is named recalculateOrder. This section of the diagram has been completed.

### *Changes to the design*

There were several changes to the design in this section, and without these changes the function would have been unsuccessful. Firstly, the accompID 'null' is not included in the array accompIDs by using an if statement. Therefore, examinees with no accompanist are not involved in calculations of exam time, available time, or free time. The function later corresponds to the design by adding examinees with no accompanist to the end of newExamineeIDs.

It also includes validation to check that the time is not 17:00. After an error was returned in unit testing, I included validation to prevent the algorithm from trying to get availability from a nonexistent field, 'Start1700'. As a result, the free time is calculated in a separate loop (which is a minor change to the design).

I have decided to make the error messages stored in \$error more helpful. By creating \$minutes\_more, and emphasising the key parts of the message, Ms Pearcey has all the data she needs to make an informed decision about how to fix the error. This change in the design makes the system more usable and helpful through improved on-screen help, meeting the usability features outlined in [2.3](#).

I have developed more functions than initially designed, to improve modularity and prevent repetition in the code. It also makes the system more maintainable. However, none of these functions have caused changes to the sequence of instructions in the flowchart in [2.5.2](#).

Although there have been many changes to the design, none have been related to the main complexity of this function. These changes are mostly to improve the system, by making it more usable or robust. This shows that my design has been quite successful and comprehensive, as it has helped me to build quite a complex function with minimal changes needed.

## *Review of the project so far*

The most difficult function has now been developed, `recalculateOrder`. This successfully achieves several success criteria, and contains a significant amount of complexity. It quantifies the idea of 'priority' into 'free time', and uses a series of loops, functions and complex SQL queries to return the desired output. This function has successfully passed all tests, and achieves exactly what it has been designed to do. This function contains almost 200 lines of code, and I have found it by far the most difficult to implement. After constant testing and improvements, I am pleased with the final iteration of this function.

This stage, as well as some previous stages, has included slightly more validation than initially expected. The function `meetingRegulations` is essentially validating the status of the schedule, which was designed previously. However, many functions have resulted in failed tests when given boundary data (such as time = 17:00). I hope to improve on this in future projects, by considering exactly how these inputs should be processed by the system during design. The next function to be developed is the `schedule` function. Although it is quite subtle, this function includes lots of validation by preventing other functions from being run with certain arguments. For example, if the array `examineeIDs` is empty, the while loop ends. This is because the schedule is complete, so it must be displayed, but it also prevents `recalculateOrder` from ever receiving an empty list. Validation such as this has not been explicitly mentioned throughout this document.

The next stage will develop the `schedule` function. This function will combine several functions together to create a valid schedule and display it. Successfully completing the next stage will complete the development of the most crucial part of this project. The last 5 stages ([3.4](#) - [3.8](#)) have created functions that work successfully individually. The `schedule` function will integrate all of these functions into a larger system, allowing them all to work together to produce a schedule. Due to rigorous testing throughout these stages, I hope that each module will be implemented smoothly into the `schedule` function. The next stage will not include a unit test, it will have a system test as all of these functions have been combined to achieve a larger, more complex system. The function 'squeeze' will be developed afterwards, as it is not nearly as crucial to the system as the other functions. It is also relatively simple to integrate this into the `schedule` function.

The directory structure has not changed, as all functions have been developed in '`schedule.php`'. This structure is shown below.

```
soundbro/
|--- index.php
|--- login.php
|--- register.php
|--- register-student.php
|--- register-supervisor.php
|--- dashboard.php
|--- schedule.php
|--- logout.php
|--- styles.css
|--- includes/
```

```
| --- db-connect.php  
| --- navbar.php  
| --- footer.html  
|--- logo/  
|   --- full.png (main logo)  
|   --- favicon3.png
```

## 3.9 The schedule function

### 3.9.1 Developing

This function, if successful, will be able to take validated data from the database and produce a schedule. This schedule will meet TCL regulations, and be created efficiently to ensure no one's time is wasted. The schedule will be created by adding records to the 'Schedule' entity in the database, and once complete it will be displayed on screen. It will also display any error messages, including error messages returned by other functions. This function will bring together many stages of development by integrating several modules into the system. I will be using the flowchart created in section 2.5.2 to help me develop this function.

```
function schedule($conn){  
    // initialise variables/classes/constants  
    $numCompletedBreaks=0;  
    $time = new DateTime('09:00');  
    $sessionTime=0;  
    $dayExamTime=0;  
    $eod=false;  
    $error="";  
    $interval = new DateInterval("PT10M");  
  
    //create examineeIDs  
    $examineeSQL="SELECT ExamineeID FROM Examinees";  
    $examineeResult=$conn->query($examineeSQL);  
    $examineeIDs = [];  
    while ($row = $examineeResult->fetch_assoc()) {  
        $examineeIDs[] = $row['ExamineeID'];  
    }  
}
```

Something I have learned from developing other modules is that \$conn must be passed as a parameter for the function to be able to access the database. Many variables will be used to keep track of the schedule, so they have been initialised at the start. I made the interval an instance of the DateInterval class, not a constant. This is because DateInterval can easily be used to increment instances of DateTime. As interval will only be used for this purpose, it is simpler to make interval a new DateInterval, as it reduces complexity in the code.

I then created the array of examineeIDs by taking them from the database. This array contains all unscheduled examinees, so for now it will contain everyone. I tested this was working correctly by outputting examineeIDs to the screen.

```
1 , 2 , 3 , 4 , 5 , 6 , 7 , 8 , 9 , 10 , 11 , 12 , 13 , 14 , 15
```

After checking the database, I was happy that this list contains all ExamineeIDs in the Examinees table. Please note that this list will usually not be sequential, as ExamineeIDs are simply a selection of StudentIDs. Now everything has been created, my main while loop can begin.

```
while (count($examineeIDs) != 0 and $eod == false and $error == ""){  
    //sort examinees by priority and other factors  
    [$examineeIDs,$miniError]=recalculateOrder($examineeIDs,$time,$conn);  
    $error.= $miniError; //append miniError to error
```

This while loop is created to schedule many accompanists, by incrementing the object \$time with each iteration. It will end if the schedule is complete, as the length of examineeIDs would be zero. It will also end if it is the end of the day (eod=True), or if an error is found. The first instruction at the start of this loop is to call 'recalculateOrder'. This will order examinees by priority, and allow them to be scheduled in the correct groups.

During my design, the only function to return an error message was recalculateOrder. As a result of additional validation being put in place, several functions will now return an error message. All of these functions will be added to one variable, \$error. The next set of instructions will iterate through examineeIDs to find an available accompanist. If found, it should break out of the loop. If not, it should keep iterating through examineeIDs.

```
//search for available accompanist  
$available=false;  
$i=0;  
while ($available == false and $i < count($examineeIDs)){  
    $available=accompAvailable($examineeIDs[$i],$time,$conn);  
    if (!$available){  
        //iterate through examineeIDs  
        $i++;  
    }  
}
```

This loop is where recalculateOrder becomes very helpful. When iterating through examineeIDs, the loop is checking for accompanists from most urgent to least urgent (as it is sorted by priority). This allows the most urgent exams to be scheduled first, as long as their accompanist is available. The function accompAvailable has been called to check if an accompanist is available for a given time. It will return true or false (available or unavailable). As I am iterating through an array, it would seem sensible to use a for loop. However, if accompAvailable returns true, the loop must end. I could use 'break' to break out of the loop if this is the case, but I try not to use this statement while programming. This is because it makes the code harder to understand, more prone to errors, and creates poorly structured code. Therefore, I used a while loop instead of a for loop. A while loop uses condition-controlled iteration, which means that I have much more control over when the loop should end. By creating a variable called 'i', and incrementing it at the end of the loop, I can iterate through examineeIDs. However, if accompAvailable returns true, 'i' will **not** be

incremented, and the loop will end. This is possible by including 2 conditions in the while loop, which is not possible using a for loop.

```
if($available){  
    // ---  
} else {  
    //increment time to next 30min slot  
    $difference=timeDifference($time);  
    $time->modify("+{$difference} minutes");  
}
```

If the loop ends and \$available is true, then 'i' will store the index of the examinee who could be scheduled next. If the loop ends and \$available is false, then no accompanists are available, so the time is incremented to the next 30 minute block. I developed the latter first. In order to increment the time to the next 30 minute slot, I had to calculate the time difference between now and the next slot. Fortunately, I have done this calculation before, when I calculated the available time for accompanists in recalculateOrder. To repeat this code would be bad programming practice, as it reduces consistency throughout the system, and makes it harder to debug or maintain. Therefore, using the code I developed in recalculateOrder, I created another function called 'timeDifference'. This returns the number of minutes until the next 30 minutes slot.

```
// calculate time until next 30 minute slot  
function timeDifference($time){  
    // get rounded time, and field corresponding to time  
    [$roundDownTime,$currentField]=getAccompField($time);  
    //adjust roundDownTime so it stores time of next field  
    $roundDownTime->modify('+30 minutes');  
  
    //find difference between current time and next 30min slot  
    $difference = $time->diff($roundDownTime);  
    $difference=$difference->i; //format so it stores number of minutes  
    return $difference;  
}
```

I also changed the code in recalculateOrder as well. By creating a single function, my solution becomes more modular and more maintainable. Once the time is incremented, the while loop will run again with this new time.

Next, I developed what to do if an available accompanist is found. The first step is to find the duration of this examinee's exam. Coincidentally, this has already been developed as a separate function, getExamineeInfo. By calling this function, I got the duration. This function also returns the AccomplID of the examinee's accompanist, but this is not used by the function. This means that the code is slightly inefficient, as it is storing a variable for no reason. However, it allows the function getExamineeInfo to run correctly, which I believe is a better, more modular structure than repeating code and making one small adjustment.

```

if($available){
    $ExamineeID=$examineeIDs[$i];

    //get duration of examinee
    [$duration,$accompID] = getExamineeInfo($ExamineeID,$conn);
}

```

Then, I created new time variables using this duration. These new time variables represent the possible end time of the schedule / start time of next exam. If these variables are breaking the regulations, a break should be scheduled. (My design includes some additional complexity in this section, by calling the squeeze function in certain situations. This function has not been developed yet, once developed it will be integrated into my solution here.) The function meetingRegulations conducts several validation checks to ensure that the proposed exam would not break any regulations. The result is either true or false (meeting or breaking regulations), which is stored in isValid.

```

//validate the possible schedule change
$newTime = $time->modify("+{$duration} minutes");
$newSessionTime = $sessionTime + $duration;
$newDayExamTime = $dayExamTime + $duration;
isValid = meetingRegulations($newTime,$newDayExamTime,$newSessionTime,$numCompletedBreaks);

[$nextBreak,$miniError]=nextBreak($numCompletedBreaks);
$error.= $miniError;
$upperBound = $nextBreak->add($interval);

```

nextBreak returns the ideal time of the next break to be scheduled. upperBound stores the latest time when an exam can end. I made interval an instance of DateInterval instead of a constant, as it makes the process of creating upperBound very simple. I also append any error messages to \$error. If there is no error message, an empty string is appended which does not change the contents of \$error. The next if/else statement will decide whether to schedule an exam or a break. Once developed, the squeeze function will be integrated into the 'else' block, just as designed in the flowchart.

```

if ($newTime <= $upperBound and $isValid){
    $StartTime=$time;
    // insert into schedule
    // remove examinee from examineeIDs
    // update time variables
} else {
    [$time,$sessionTime,$numCompletedBreaks,$eod,$miniError]=addBreak($time,$numCompletedBreaks);
    $error.= $miniError;
}

```

newTime stores time + duration, so it stores the end time of the proposed exam. If the exam ends before the upper bound of the next break, **and** meetingRegulations returns true, then the proposed exam is validated. Therefore, this exam is scheduled. If not, then a break must be scheduled by calling addBreak. This function will schedule a break by updating the time

variables, and also determine if it is the end of the day. Next I developed the functionality of scheduling an exam.

```
if ($newTime <= $upperBound and $isValid){

    // insert into schedule
    $StartTime = $time;
    $schedulingSQL = "INSERT INTO Schedule (StartTime, ExamineeID) VALUES ($StartTime, $ExamineeID)";
    $conn->query($schedulingSQL);

    // remove examinee from examineeIDs
    unset($examineeIDs[$i]);

    // update time variables
    $time = $newTime;
    $sessionTime = $newSessionTime;
    $dayExamTime = $newDayExamTime;
}
```

First, I inserted the relevant data into the Schedule table. Once inserted, the examinee has been scheduled. Therefore, they are removed from examineeIDs. Now that the exam has been scheduled, time is set to newTime as it needs to be incremented to the start time of the next possible exam. This allows the function to run the while loop again, scheduling exams/breaks and incrementing the time until the loop is broken. The function should now be able to schedule exams by populating the Schedule table in the database.

The final step is to output the relevant messages, and the schedule. I have included an additional variable, \$alert, which will contain important information about the process.

```
$alert = "";
if ($eod){
    //display end of day message
    $alert .= "The day has ended <br> ";
}
if (count($examineeIDs) != 0){
    // display unscheduled examinees
    $alert .= "The following examinees have not been scheduled: ".implode(", ", $examineeIDs). "<br>";
} else {
    $alert .= "All examinees have been scheduled!";
}
```

\$alert will display the messages shown above, to give the most relevant information to Ms Pearcey. This allows her to develop a deeper understanding of the scheduling process, and gives the relevant information needed for her to adjust the inputs. The \$alert and \$error will not be displayed, instead they will be returned to the main program. This gives me more control over how to display it, so that the message is clear and direct.

```

if ($error=="") {
    //display schedule
    $getSchedule = "SELECT * FROM Schedule";
    $result = $conn->query($getSchedule);
    // Check if schedule is empty
    if ($result->num_rows > 0) {
        while ($row = $result->fetch_assoc()) {
            echo "ScheduleID: ".$row["ScheduleID"] .
                " | StartTime: " . $row["StartTime"] .
                " | ExamineeID: " . $row["ExamineeID"] . "<br>";
        }
    } else {
        $alert.= "Schedule is empty";
    }
}
return [$alert,$error];

```

As long as there are no errors, the schedule will be displayed. The function is now complete, bringing all functions from previous stages together. I will conduct system testing to test how this process is working.

### 3.9.2 System test of the schedule function

In previous sections, I have conducted several unit tests of smaller functions. I am now in the position to conduct a full system test, with all functions integrated into the system. This ensures that the functions will work together to achieve the expected results. I will use the fake data in the database for these tests, as it gives me the flexibility to adjust data to test all functionality in the system. The first test is simply to run the function. I expect to receive no error messages.

Purpose of test	Test data	Expected Result	Successful?
Testing that the schedule function produces a full schedule	Using fake data in the database	Full schedule produced	No

( ! ) Warning: Undefined variable \$currentField in C:\wamp64\www\soundbro\schedule.php on line 255

Call Stack

The variable \$currentField has not been recognised. This is within the function recalculateOrder, when trying to calculate the available time for each accompanist. As a result of extensive unit testing in this section, I found it surprising that there was an error here. However, I have edited this code in this stage, when I created the function timeDifference. Previously, recalculateOrder was working fine, but now that this function is being called instead, there is an error. I noticed that currentField is being created in timeDifference and not being returned. Therefore, the variable currentField has not been defined in recalculateOrder. The simple solution to this is to return this variable to the function, but I must be careful to store the second variable in **all** locations where timeDifference is called.

```
return [$difference,$currentField];
```

```
// calculate time until next 30min slot
[$difference,$currentField]=timeDifference($time);
```

This should fix the issue, so I will run the test again.

Purpose of test	Test data	Expected Result	Successful?
Testing that the schedule function produces a full schedule	Using fake data in the database	Full schedule produced	No



Fatal error: Uncaught Error: Object of class DateTime could not be converted to string in C:\wamp64\www\soundbro\schedule.php on line 448

This error states that I am trying to convert an object to string. Line 448 is shown below.

```
$schedulingSQL="INSERT INTO Schedule (StartTime,ExamineeID) VALUES ($StartTime,$ExamineeID);
```

\$StartTime is an object, so cannot be inputted into the table. I must format it into HH:MM for it to be accepted

```
// insert into schedule
$StartTime=$time->format('H:i');
$schedulingSQL="INSERT INTO Schedule (StartTime,ExamineeID) VALUES ($StartTime,$ExamineeID)";
$conn->query($schedulingSQL);
```

I have now formatted the time appropriately. H indicates hours, and i indicates minutes, so \$StartTime should now be a string in the form 'HH:MM'. This should fix the error, so I will run this test another time.

Purpose of test	Test data	Expected Result	Successful?
Testing that the schedule function produces a full schedule	Using fake data in the database	Full schedule produced	No



Warning: Undefined array key 6 in C:\wamp64\www\soundbro\schedule.php on line 206

Call Stack



Fatal error: Uncaught Error: Call to a member function fetch\_assoc() on bool in C:\wamp64\www\soundbro\schedule.php on line 315



Error: Call to a member function fetch\_assoc() on bool in C:\wamp64\www\soundbro\schedule.php on line 315

I was unsure exactly how to solve this issue, so I tried some debugging techniques. I noticed some other errors while doing this. Firstly, the object \$time was being modified whenever I created newTime. Therefore, I adjusted the code so that time is only modified after scheduling.

```
$newTime= clone $time;
$newTime->modify("+$duration} minutes");
```

After some research, I found that the lower error may occur if I don't put single quotes around variable names in the SQL. I adjusted this for SQL queries throughout the program. This removed the second error, but the first still existed. After printing several different variables to the screen, I realised that the array key 6 was not found because it was being removed from the array (it was being scheduled). For clarification, my array of examineeIDs is shown below after recalculating the order once.

([0] => 6 [1] => 4 [2] => 15 [3] => 1 [4] => 3 [5] => 10 [6] => 12 [7] => 7 [8] => 13 [9] => 9 [10] => 8 [11] => 2 [12] => 11 [13] => 5 [14] => 14)

Then, ExamineeID 12, or array key 6, is scheduled. Therefore, they are removed from the array. This is left behind.

([0] => 6 [1] => 4 [2] => 15 [3] => 1 [4] => 3 [5] => 10 [7] => 7 [8] => 13 [9] => 9 [10] => 8 [11] => 2 [12] => 11 [13] => 5 [14] => 14)

When the while loop runs again, and it tries to iterate through the list, it cannot find array key 6. To solve this issue, I need the array keys to become sequential after every time an examinee is removed. The function array\_values does this process.

```
// remove examinee from examineeIDs
unset($examineeIDs[$i]);
$examineeIDs=array_values($examineeIDs);
```

Now, the array after removing key 6 is shown below.

```
( [0] => 6 [1] => 4 [2] => 15 [3] => 1 [4] => 3 [5] => 10 [6] => 7 [7] => 13 [8] => 9 [9] => 8 [10]
=> 2 [11] => 11 [12] => 5 [13] => 14 )
```

The array keys are sequential, so all keys have been defined from 0 to count(\$examineeIDs)-1.

Purpose of test	Test data	Expected Result	Successful?
Testing that the schedule function produces a full schedule	Using fake data in the database	Full schedule produced	Yes! (see below)

```
ScheduleID: 1 | StartTime: 09:00:00 | ExamineeID: 12
ScheduleID: 2 | StartTime: 09:13:00 | ExamineeID: 7
ScheduleID: 3 | StartTime: 09:31:00 | ExamineeID: 13
ScheduleID: 4 | StartTime: 09:49:00 | ExamineeID: 9
ScheduleID: 5 | StartTime: 10:02:00 | ExamineeID: 8
ScheduleID: 6 | StartTime: 10:30:00 | ExamineeID: 6
ScheduleID: 7 | StartTime: 10:58:00 | ExamineeID: 4
ScheduleID: 8 | StartTime: 11:11:00 | ExamineeID: 15
ScheduleID: 9 | StartTime: 11:24:00 | ExamineeID: 2
ScheduleID: 10 | StartTime: 11:37:00 | ExamineeID: 1
ScheduleID: 11 | StartTime: 11:55:00 | ExamineeID: 3
ScheduleID: 12 | StartTime: 12:18:00 | ExamineeID: 10
ScheduleID: 13 | StartTime: 13:36:00 | ExamineeID: 5
ScheduleID: 14 | StartTime: 13:59:00 | ExamineeID: 14
ScheduleID: 15 | StartTime: 14:30:00 | ExamineeID: 11
```

All examinees have been scheduled!

This is a great sign, and shows that the schedule is being created correctly. Before I go any further, there is more to be done in displaying the schedule. The function displaySchedule will correctly do this, but I will develop a basic prototype of this solution for now. Currently, the schedule cannot be understood by anyone, so I must retrieve the relevant information from the database to produce an informative timetable in the format shown below.

Examinee Name	Instrument	Grade	Accompanist Name	Start Time
---------------	------------	-------	------------------	------------

The final PDF of the schedule will be of the same format. During my design, I confirmed this style with Ms Pearcey, to ensure that she was happy with the level of information provided. She was happy with this design, as the email below shows.

Rohan Desai <17desai407@camphillboys.bham.sch.uk> 17:53 (4 hours ago)

to Lorne ▾

Hi miss,

Hope you are doing well. I have a quick question: what should I put in the final PDF of the schedule? Currently I am thinking of putting the student's first and last name, instrument, grade, accompanist and start time. Is there anything else to put in?

Thanks,  
Rohan

---

Lorne Pearcey 18:33 (3 hours ago)

to me ▾

No. I think that's fine.

Ms L. Pearcey

(repeated) Figure 2.2.1h: Checking requirements of the PDF file with my client.

It is quite complex to achieve a well-formatted table through just using 'echo'. Therefore, I will be redirecting the user to another page. Additionally, I have decided to output the schedule regardless of what error is displayed. I believe it is important to give Ms Pearcey the full information, and it is unnecessary to abstract this from her. Without the full story, she could find it more difficult to make the changes needed to fix these errors. The schedule will now return \$alert and \$error, and the main program will take the user to 'schedule-complete.php'. \$alert and \$error will be converted to session variables, and displayed on the new page.

```
// schedule exam on button click
if (isset($_POST['schedule'])) {
    [$alert,$error]=schedule($conn);
    //redirect to schedule-complete.php with messages
    $_SESSION['alert']=$alert;
    $_SESSION['error']=$error;
    header("Location:schedule-complete.php");
    exit();
}
```

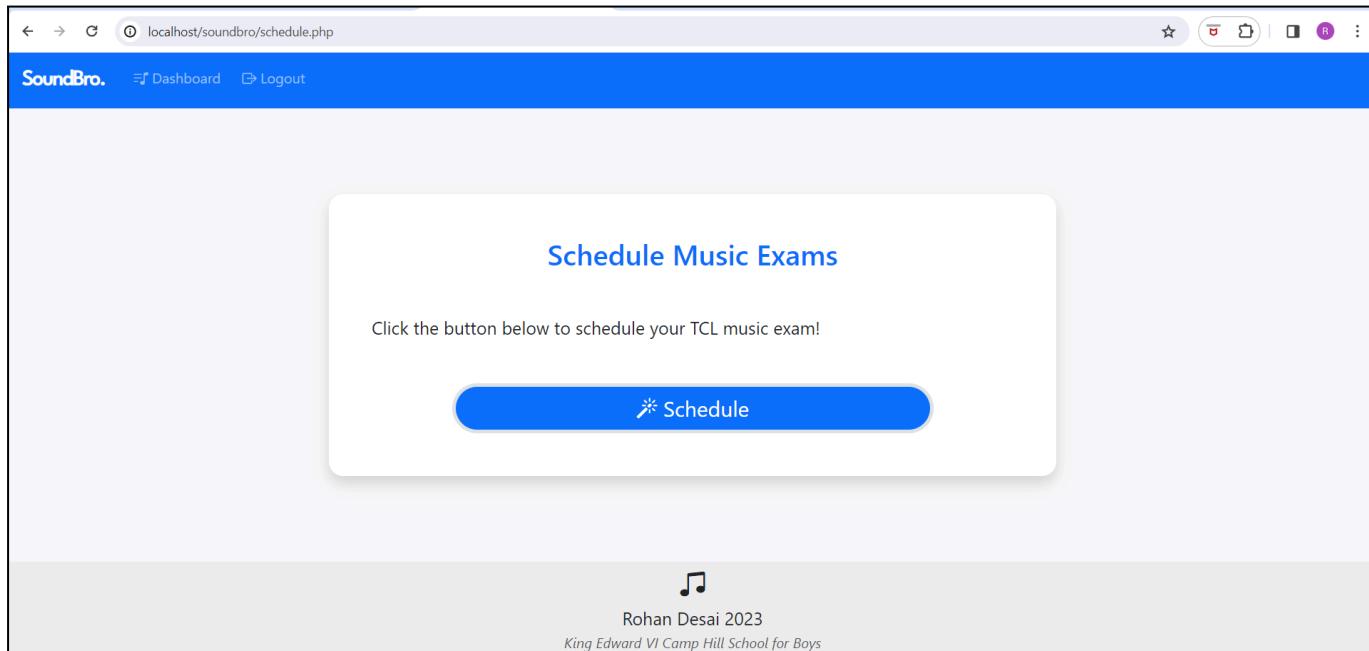
Upon clicking a button 'schedule', the schedule will be created and the user will be taken to schedule-complete.php. I have also used TRUNCATE to empty the Schedule table at the start of this function. This prevents schedules being repeated multiple times in the table.

```

function schedule($conn){
    $emptySchedule="TRUNCATE TABLE Schedule";
    $conn->query($emptySchedule);
}

```

Below is a simple prototype of the interface for schedule.php.



Using Bootstrap 5 icons, I have created a clear button with a memorable icon. The magic wand illustrates the seemingly magical process of scheduling all exams. It is large, highlighted in the primary colour, and easy to press. As stated in [2.3 Usability features](#), it is important that this button is clearly emphasised. I feel that this makes the system easier to use, and achieves one of the key usability features designed in section 2.3. By abstracting all unnecessary information from the screen, the user will not feel overwhelmed. This prevents the interface from looking like [1.4.3 Furlong Maestro](#), an existing solution with a confusing and inelegant interface.

Currently, clicking schedule will create the schedule and redirect to schedule-complete.php. The error and alert messages will be displayed here. I will use a red background to show errors, to indicate that there is an issue. I will use light blue for the alert, simply to emphasise the message. I need to now develop the interface for the table. However, this is more complex than displaying a table as before. I need to make SQL queries to retrieve all relevant information, so that the table is informative and easy to interpret. It shouldn't contain any IDs, as these are useless to anyone that is not familiar with the database.

```
// get info from DB to create correct schedule format
$scheduleInfoSQL = "SELECT ScheduleID,FirstName,LastName,Instrument,Grade,StartTime,AccompID
FROM students
JOIN examinees ON students.StudentID = examinees.ExamineeID
JOIN schedule ON examinees.ExamineeID = schedule.ExamineeID;";
```

This query will get most of the data needed from the database. I tested this query works using phpMyAdmin's interface

Showing rows 0 - 13 (14 total, Query took 0.0009 seconds.)

```
SELECT ScheduleID,FirstName,LastName,Instrument,Grade,StartTime,AccompID FROM students JOIN examinees ON students.StudentID = examinees.ExamineeID JOIN schedule ON examinees.ExamineeID = schedule.ExamineeID;
```

Profiling [ Edit inline ] [ Edit ] [ Explain SQL ] [ Create PHP code ] [ Refresh ]

ScheduleID	FirstName	LastName	Instrument	Grade	StartTime	AccompID
1	Sophia	Hall	trombone	2	09:00:00	2
2	Daniel	Williams	trombone	5	09:13:00	2
3	Noah	Turner	trombone	5	09:31:00	2
4	Ethan	Clark	trumpet	1	09:49:00	2
5	Emma	Davis	french horn	1	10:02:00	1
6	Emily	Miller	classical guitar	3	10:15:00	1
7	Logan	Wilson	classical guitar	3	10:28:00	1
9	Olivia	Jones	trumpet	8	11:14:00	2
10	Jane	Smith	violin	2	11:42:00	2
11	Michael	Brown	trumpet	7	11:55:00	NULL
12	Alex	Johnson	classical guitar	6	13:18:00	1
13	Ava	Moore	violin	4	13:41:00	1
14	Isabella	Scott	classical guitar	7	13:59:00	NULL
15	Liam	White	violin	6	14:30:00	2

Therefore, I can trust that this query will return the necessary information correctly. Using echo, I output each piece of information using the <td> tag, except for the start time and accompanist name. As some examinees have no accompanist, I cannot always display a name. If I try to return a name when AccompID = null, I will get an error. Therefore, I have added some additional validation by checking that the AccompID exists. If it does, the following code is executed.

```

$AccompID=$row['AccompID'];
if ($AccompID){
    // if examinee has accompanist, display their name
    $scheduleID=$row['ScheduleID'];
    $accompInfoSQL="SELECT FirstName,LastName
    FROM Supervisors
    JOIN ExamAccomps ON Supervisors.SupervisorID = ExamAccomps.AccompID
    WHERE AccompID = '$AccompID'";
    $accompInfo=$conn->query($accompInfoSQL);
    $accompRow=$accompInfo->fetch_assoc();
    echo "<td>{$accompRow['FirstName']} {$accompRow['LastName']}

```

If AccompID is null, the code will output an empty cell. This ensures that the start time is always in the same row. Finally, the start time is outputted. I have used the  **tag to emphasise it, as it is the most important part of this schedule. Now that the interface for displaying the schedule is complete, I can run the same test again.**

Purpose of test	Test data	Expected Result	Successful?
Testing that the schedule function produces a full schedule	Using fake data in the database	Full schedule produced	Yes (See below)

## Scheduling Results

All examinees have been scheduled!

### Exam Schedule

Student Name	Instrument	Grade	Accompanist Name	Start Time
Sophia Hall	trombone	2	Sarah Johnson	09:00:00
Daniel Williams	trombone	5	Sarah Johnson	09:13:00
Noah Turner	trombone	5	Sarah Johnson	09:31:00
Ethan Clark	trumpet	1	Sarah Johnson	09:49:00
Emma Davis	french horn	1	David Williams	10:02:00
Emily Miller	classical guitar	3	David Williams	10:15:00
Logan Wilson	classical guitar	3	David Williams	10:28:00
Joe Davies	classical guitar	4	David Williams	10:56:00
Olivia Jones	trumpet	8	Sarah Johnson	11:14:00

Jane Smith	violin	2	Sarah Johnson	11:42:00
Michael Brown	trumpet	7		11:55:00
Alex Johnson	classical guitar	6	David Williams	13:18:00
Ava Moore	violin	4	David Williams	13:41:00
Isabella Scott	classical guitar	7		13:59:00
Liam White	violin	6	Sarah Johnson	14:30:00



Rohan Desai 2023  
King Edward VI Camp Hill School for Boys

As you can see, the interface is clear and emphasises the start time. It displays all relevant information, as me and Ms Pearcey agreed on. However, there are 2 improvements to be made which would make the output more useful.

- To emphasise when the breaks are, there should be an empty row.
- The start time should abstract the number of seconds, as it is irrelevant.

As the time is always in the form HH:MM:SS, the first 5 characters of the string should be included. Therefore, I can extract this portion of the string using the substr() function.

```
$StartTime=substr($row['StartTime'],0,5);
echo "<td><strong>{$StartTime}</strong></td>";
```

Start Time
09:00
09:13
09:31

The start time is now being displayed in the most efficient and direct way. In order to emphasise when the breaks are, I need to keep a running count of the time as the loop iterates through each record in the schedule.

```
$time= new DateTime('09:00');
while ($row = $scheduleInfo->fetch_assoc()) {
```

Once in the loop, I wrote code to get the duration of an examinee's exam. The \$time is incremented by the duration so that it stays on track with the current exam. This happens at the **end** of the while loop.

```
//increment time
$time->modify("+{$duration} minutes");
```

The start time of the current exam should match the object \$time. If it doesn't, then the end time of the previous exam is not equal to the start time of the current exam. In this situation, a break has been scheduled, so the table displays an empty row.

```
// if times dont match, add an empty row
if ($time->format('H:i') != $StartTime){
    // empty row to signal a break
    echo "<tr><td colspan='5'><br></td></tr>";
    // reset time
    $StartTime_object = DateTime::createFromFormat('H:i', $StartTime);
    $time->setTime($StartTime_object->format('H'), $StartTime_object->format('i'));
}
```

Once complete, the \$time must reset so that it is equal to the current exam's start time. This syncs them back up again. Using the 'createFromFormat' method, I created StartTime\_object. \$time is then set equal to StartTime\_object. This means that the next block of exams will be in sync again, so the next time they are out of sync, I can be sure that another break has occurred. This running object of time allows me to determine when breaks occur, so that empty rows can be outputted to clearly show the different sessions.

To make the schedule appear more formal, I have also capitalised the instrument name using the function ucwords(). Once again, I will run this test to assess if the system is working correctly.

Purpose of test	Test data	Expected Result	Successful?
Testing that the schedule function produces a full schedule	Using fake data in the database	Full schedule produced	Yes

### Exam Schedule

Student Name	Instrument	Grade	Accompanist Name	Start Time
Sophia Hall	Trombone	2	Sarah Johnson	09:00
Daniel Williams	Trombone	5	Sarah Johnson	09:13
Noah Turner	Trombone	5	Sarah Johnson	09:31
Ethan Clark	Trumpet	1	Sarah Johnson	09:49
Emma Davis	French Horn	1	David Williams	10:02
Emily Miller	Classical Guitar	3	David Williams	10:15
Logan Wilson	Classical Guitar	3	David Williams	10:28
Joe Davies	Classical Guitar	4	David Williams	10:56
Olivia Jones	Trumpet	8	Sarah Johnson	11:14
Jane Smith	Violin	2	Sarah Johnson	11:42
Michael Brown	Trumpet	7		11:55
Alex Johnson	Classical Guitar	6	David Williams	13:18
Ava Moore	Violin	4	David Williams	13:41
Isabella Scott	Classical Guitar	7		13:59
Liam White	Violin	6	Sarah Johnson	14:30

I am very happy with how this schedule is being displayed. I find it clear and informative, while not being overwhelming. It abstracts all unnecessary detail so that the viewer can easily understand everything on screen. This is a good solution, but just a prototype. In future stages I will develop the displaySchedule function to make the process of displaying schedules much more efficient and modular. Some changes made here (such as splitting the schedule into sessions) will also be made when developing displaySchedule.

Moving on to the actual schedule, it is mostly successful. The schedule has clearly tried to group accompanists into blocks, but due to limited availability from Sarah Johnson, they are split up throughout the day. This is completely fine, as the availability of accompanists should be strict. All 15 examinees have been scheduled. Within each accompanist block (especially

the first 2), exams are grouped by instrument family, then instrument, then sorted ascending by grade. Breaks are added at good times, as they are quite close to the ideal break times. The schedule meets TCL regulations, as session time never exceeds 2 hours, day exam time is less than 6.5 hours, a lunch break is scheduled before 12:45, and the timetable ends before 5pm.

Logan Wilson's exam is 13 minutes, so it ends at 10:41. At this time, a 15 minute break is scheduled. This is 4 minutes off the ideal break time, which is perfectly okay. Michael Brown has a 23 minute exam which ends at 12:18. A 1 hour lunch break is then scheduled, which is 12 minutes earlier than the ideal time. This is okay as well.

Isabella Scott's exam ends at 14:22. The last examinee to be scheduled is Liam White, but his accompanist is not available at this time. Therefore, there is a pause for 8 minutes, and then he is scheduled. This, again, is a completely valid decision. As Sarah Johnson has much less availability, she gets the most priority and is playing exams during almost all available time. However, this is still not enough, so a short pause is scheduled until she is available.

While this is all valid, I realised that there is one potential issue in my code, that could lead to invalid schedules being produced. If an accompanist is rarely available, the time may have to be incremented to the next 30 minute slot (as seen above). The code that increments the time in this way will **never** check if it is breaking regulations or not. This can lead to many errors, as the time could be incremented in such a way that invalid timetables are created. In the next test, I adjusted the availability so that no one is available from 12 to 1.

Purpose of test	Test data	Expected Result	Successful?
Testing how the schedule responds to extended unavailability from accompanists	Using fake data in the database  (no accompanists are available from 12 to 1, but they are from 1:30)	Schedule produced with all examinees, minimal time wasted.	No

## Exam Schedule

Student Name	Instrument	Grade	Accompanist Name	Start Time
Sophia Hall	Trombone	2	Sarah Johnson	09:00
Daniel Williams	Trombone	5	Sarah Johnson	09:13
Emma Davis	French Horn	1	David Williams	09:31
Noah Turner	Trombone	5	Sarah Johnson	09:44
Emily Miller	Classical Guitar	3	David Williams	10:02
Ethan Clark	Trumpet	1	Sarah Johnson	10:15
Logan Wilson	Classical Guitar	3	David Williams	10:28
Olivia Jones	Trumpet	8	Sarah Johnson	10:56
Jane Smith	Violin	2	Sarah Johnson	11:24
Michael Brown	Trumpet	7		11:37
Isabella Scott	Classical Guitar	7		12:00
Joe Davies	Classical Guitar	4	David Williams	14:00
Alex Johnson	Classical Guitar	6	David Williams	14:18
Ava Moore	Violin	4	David Williams	14:41
Liam White	Violin	6	Sarah Johnson	14:59

This timetable shows a concerningly large gap in the schedule. Isabella Scott's exam ends at 12:23. Then, a 1 hour and 37 minute exam is scheduled! This is longer than necessary, and breaks the TCL regulations. A lunch break **must** be scheduled before 12:45. At 12:23, no accompanists are available for a full exam (because both accompanists are unavailable from 12 to 1), and there are no more examinees with no accompanists. Therefore, the schedule skips ahead to 12:30. At this time, it checks if anyone is available. Once again, no one is available, so it skips ahead at 13:00. Now, at this time the schedule has found an available accompanist, but according to the algorithm it needs to schedule a lunch break. Therefore, a full hour is scheduled for a lunch break, even though over 30 minutes have already passed. To solve this issue, the algorithm must check if the time is within an interval of the next break. It also needs to check this schedule against the regulations, this is something that should be done whenever a change is made to the schedule. Without validating the schedule using meetingRegulations, errors like this occur.

The change should make this incrementing process very similar to the process of scheduling an actual exam. All changes will happen inside the 'else' block of the following if statement

```
if($available){
```

First, the time of the next break is calculated by calling nextBreak. Using this time and the interval, the upper and lower bounds for the ideal break time are calculated.

```

} else {
    //find bounds for ideal break time
    [$nextBreak,$miniError]=nextBreak($numCompletedBreaks);
    $error.= $miniError;
    $upperBound = $nextBreak->add($interval);
    $lowerBound = $nextBreak->sub($interval);

```

If the current time is within these bounds, it makes sense to schedule a break (as it is pretty close to the ideal break time).

```

//can a break be scheduled now?
if ($time >= $lowerBound and $time <= $upperBound){
    [$time,$sessionTime,$numCompletedBreaks,$eod,$miniError]=addBreak($time,$numCompletedBreaks);
    $error.= $miniError;

```

If the current time is outside of these bounds, the time may be incremented to the next 30 minute slot. But before this is done, meetingRegulations must check that this proposed change is valid. Note that \$difference is the number of minutes until the next 30 minute slot.

```

//validate the possible schedule change
$newTime= clone $time;
$newTime->modify("+$difference minutes");
$newSessionTime = $sessionTime + $difference;
isValid = meetingRegulations($newTime,$dayExamTime,$newSessionTime,$numCompletedBreaks);

```

If the time is incremented, the variable time should be updated, as should sessionTime. However, dayExamTime should not be changed, as we are not scheduling an exam, we are simply incrementing the time.

```

if ($isValid){
    //if valid, increment time to next 30min slot
    $time=$newTime;
    $sessionTime=$newSessionTime;
} else {
    //if invalid, schedule a break
    [$time,$sessionTime,$numCompletedBreaks,$eod,$miniError]=addBreak($time,$numCompletedBreaks);
    $error.= $miniError;
}

```

Once the proposed changes are validated, the time can be incremented to the next 30 minute slot. However, if invalid then a break is scheduled. Remember that the solution to an invalid schedule is **always** to go back and schedule a break instead, as all TCL rules are based around the timing of breaks. Now, there should be no possible route that this algorithm can take where a change is not validated. This should effectively prevent invalid schedules from being produced, so I will run the test again to check.

**Please note:** The schedule produced above also contained an unexpected ‘alternating’ pattern between accompanists. In the morning, exams for David Williams and Sarah Johnson alternated, instead of being grouped. This issue has been addressed later in the testing.

Purpose of test	Test data	Expected Result	Successful?
Testing how the schedule responds to extended unavailability from accompanists	Using fake data in the database  (no accompanists are available from 12 to 1, but they are from 1:30)	Schedule produced with all examinees, minimal time wasted.	Yes

### Exam Schedule

Student Name	Instrument	Grade	Accompanist Name	Start Time
Sophia Hall	Trombone	2	Sarah Johnson	09:00
Daniel Williams	Trombone	5	Sarah Johnson	09:13
Emma Davis	French Horn	1	David Williams	09:31
Noah Turner	Trombone	5	Sarah Johnson	09:44
Emily Miller	Classical Guitar	3	David Williams	10:02
Ethan Clark	Trumpet	1	Sarah Johnson	10:15
Logan Wilson	Classical Guitar	3	David Williams	10:28
<hr/>				
Olivia Jones	Trumpet	8	Sarah Johnson	10:56
Jane Smith	Violin	2	Sarah Johnson	11:24
Michael Brown	Trumpet	7		11:37
Isabella Scott	Classical Guitar	7		12:00
<hr/>				
Joe Davies	Classical Guitar	4	David Williams	13:30
Alex Johnson	Classical Guitar	6	David Williams	13:48
Ava Moore	Violin	4	David Williams	14:11
<hr/>				
Liam White	Violin	6	Sarah Johnson	14:30

As Isabella Scott’s exam ends at 12:23, a 1 hour and 7 minute lunch break is scheduled. Examinees are scheduled as soon as an accompanist becomes available, which saves time and improves the efficiency of the schedule. This test is successful.

I am now happy that my algorithm can schedule exams and breaks correctly. I am also confident that it can respond well to poor accompanist availability by incrementing the time. In order to test the limits of this system, I added more examinees to the Examinees table.

Purpose of test	Test data	Expected Result	Successful?
Testing how the program responds to too many examinees being inputted	Using fake data in the database	<p>End of day message and list of unscheduled examinees</p> <p>Schedule displayed with line breaks whenever there is a break.</p>	Yes (see below)

The day has ended.  
The following examinees have not been scheduled: 14 , 10 , 11

## Exam Schedule

Student Name	Instrument	Grade	Accompanist Name	Start Time
Sophie Hill	Violin	1	Christopher Smith	09:00
Sophia Hall	Trombone	2	Sarah Johnson	09:13
Chloe Harrison	Jazz Saxophone	1	Christopher Smith	09:26
Daniel Williams	Trombone	5	Sarah Johnson	09:39
Carter Evans	Jazz Saxophone	5	Christopher Smith	09:57
Noah Turner	Trombone	5	Sarah Johnson	10:15
Olivia Jones	Trumpet	8	Sarah Johnson	10:48
Emma Davis	French Horn	1	David Williams	11:16
Mia Baker	Trumpet	5	David Williams	11:29
Emily Miller	Classical Guitar	3	David Williams	11:47
Logan Wilson	Classical Guitar	3	David Williams	12:00
Joe Davies	Classical Guitar	4	David Williams	12:13

Alex Johnson	Classical Guitar	6	David Williams	13:31
Lucas Ross	Viola	3	David Williams	13:54
Lily Morris	Clarinet	1	David Williams	14:07
Ethan Clark	Trumpet	1		14:20
Jane Smith	Violin	2	Sarah Johnson	14:33
Avery Cooper	Violin	5	Sarah Johnson	14:46
Henry Reed	Clarinet	5	Sarah Johnson	15:04
Michael Brown	Trumpet	7		15:37
Madison Barnes	Classical Guitar	1		16:00
Aiden Fisher	Classical Guitar	3		16:13
Jackson Ward	Classical Guitar	4		16:26
Grace Kelly	Jazz Saxophone	3		16:44

This exam has scheduled 24 of the 27 examinees given. This is good, as it shows that the schedule will be able to produce a valid schedule even if there are too many inputs, which shows robustness in the system.

The following breaks have been scheduled: 10:33-10:48, 12:31-13:31, and 15:22-15:37. These are 12, 1 and 7 minutes off the ideal break times (respectively). Accompanists are mostly grouped into blocks, and each block has been sorted correctly by family, instrument and grade. No session is longer than 2 hours, a lunch break is scheduled before 12:45, and the total duration of all exams is 387 minutes (3 minutes below the limit). The last exam ends at 16:57 (the limit is 17:00). The duration of all exams is correct, as per the TCL regulations on exam timings. All accompanists are only scheduled to play during the times they are available for. No exams run over into their 'unavailable' times.

The above paragraph shows clearly that this timetable meets all of the regulations for TCL exam scheduling. The schedule only has 3 more minutes before reaching the 17:00 limit, which shows that exams have been scheduled as efficiently as possible. However, there are 2 possible improvements to be made:

1. In the end of day message, the ExamineeIDs are displayed. This means nothing to the user, so their name should be displayed instead. This makes the system more helpful and usable.
2. In the first session, exams alternate between Christopher Smith and Sarah Johnson. They should be grouped together, and not alternate. The same error can be found in above tests as well. I would like to investigate why this is occurring, and try to find a solution.

In order to fix the first issue, I made a for loop that iterates through each value in examineeIDs. For each ID, I made an SQL query to get the first and last name of the current student, and added these names to an array. Once the loop is complete, the entire array is added to the alert message, with commas separating each name.

```
if (count($examineeIDs) != 0){
    // display unscheduled examinee names
    $names=[];
    for ($i=0 ; $i<count($examineeIDs) ; $i++) {
        // get examinee name
        $getNames = "SELECT FirstName, LastName FROM Students WHERE StudentID='".$examineeIDs[$i].'";
        $nameResult = $conn->query($getNames);
        $nameRow=$nameResult->fetch_assoc();
        $fname=$nameRow['FirstName'];
        $lname=$nameRow['LastName'];
        //add name to array
        $names[]=$fname . $lname;
    }
    //add all names to alert message
    $alert.= "The following examinees have not been scheduled: ".implode(" , ",$names)."<br>";
}
```

This should solve the first issue. As for the second issue, I was quite confused as to why this alternating pattern was produced. I ran some more tests with different sets of data to help me understand the issue. Once enough tests had been completed, I noticed that this was mainly occurring when 2 accompanists had similar availability. This made me reevaluate my scheduling algorithm to gain some more insight. In the cases where the alternating pattern was seen, both accompanists were available during the entire time. For example, both Sarah Johnson and Christopher Smith are available from 09:00 to 12:00 in the above example. Therefore, accompAvailable must be returning true on the very first iteration of the loop. This means that the fault lies within the function recalculateOrder.

This function will recalculate the order of examinees whenever an exam is scheduled to make sure that the most urgent exams can be scheduled first. This ‘urgency’ of an examinee’s exam is defined by their accompanist’s free time, where free time = available time - remaining exam time. Examinees are sorted ascending by free time.

Whenever an exam is scheduled, the accompanist's remaining exam time will decrease, but so will their available time. Therefore, their free time won't change. However, if another accompanist was available at the same time, and they were not scheduled, their available time will decrease, and their exam time will stay constant. This causes their free time to decrease, which makes their exams more urgent. This principle means that the most important exams are scheduled first, which maximises the number of examinees scheduled altogether.

However, this is also causing the issue shown above. Sarah Johnson and Christopher Smith both have similar free times, and similar availability. Therefore, when one is scheduled, the other is not. This causes the other person's free time to fall, and fall **below** the free time of the first accompanist. When recalculateOrder is run again, the other person now has the least free time, so they have a higher urgency and are put first in the schedule. This creates an alternating cycle, as the accompanist that is not scheduled will always be given priority for the next exam.

To illustrate my point clearly, I have outputted the array AccomplIDs each time the order is recalculated. The index 0 represents AccomplID. AccomplID 2 is Sarah Johnson, and AccomplID 3 is Christopher Smith. The index 3 represents free time.

<code>array (size=4) 0 =&gt; string '3' 1 =&gt; int 44 2 =&gt; int 240 3 =&gt; int 196 =&gt; array (size=4) 0 =&gt; string '2' 1 =&gt; int 126 2 =&gt; int 330 3 =&gt; int 204</code>	<code>array (size=4) 0 =&gt; string '2' 1 =&gt; int 126 2 =&gt; int 317 3 =&gt; int 191 =&gt; array (size=4) 0 =&gt; string '3' 1 =&gt; int 31 2 =&gt; int 214 3 =&gt; int 183 =&gt; array (size=4) 0 =&gt; string '2' 1 =&gt; int 31 2 =&gt; int 227 3 =&gt; int 196 =&gt; array (size=4) 0 =&gt; string '3' 1 =&gt; int 113 2 =&gt; int 291 3 =&gt; int 178 =&gt; array (size=4) 0 =&gt; string '2' 1 =&gt; int 113 2 =&gt; int 304 3 =&gt; int 191 =&gt; array (size=4) 0 =&gt; string '3' 1 =&gt; int 18 2 =&gt; int 201 3 =&gt; int 183 =&gt; array (size=4) 0 =&gt; string '2' 1 =&gt; int 95 2 =&gt; int 273 3 =&gt; int 178</code>
---	--

From left to right, each screenshot is the next iteration of the while loop. So this output is displayed each time an exam is scheduled. Notice how the AccomplID's 2 and 3 swap with each iteration. The index 3 (representing free time) provides more insight into this. Firstly, 196 is less than 204, so AccomplID 3 is sorted earlier and their exam is scheduled. This causes AccomplID 2's free time to drop to 191, which is lower than 196. Therefore, AccomplID 2 gets priority for the second exam. This causes AccomplID 3's free time to drop to 183, which is below 191. The cycle is repeated. This repeats until all of AccomplID 3's exams are scheduled. Therefore, when free time is similar, and both accompanists are available, an alternating pattern can be created.

Now I understand the purpose of the error, I can try to tackle it. I do not want to make any significant changes to the system, as my schedule overall is quite successful. However, I would like to tweak some aspects of my code to see if any improvements can be made. The function 'freeTimeSort' is shown below. This will sort any 2 elements in AccomplIDs. When combined with the usort function, the array AccomplIDs is sorted by free time using this function.

```

function freeTimeSort($rowA, $rowB) {
    return $rowA[3] - $rowB[3];
}

```

I have explained clearly when developing this function, but this will return a number. If positive, rowA goes after rowB. If negative, rowB goes after rowA. If 0, the order is unchanged. Fundamentally, I still agree with this logic. If one accompanist has lots of free time, and another has very limited free time, the one with less free time should be scheduled first. This optimises each accompanist's availability, and makes the algorithm able to adapt to many different combinations of availability. However, if free time is sufficiently large, we could be more lenient. In the situation above, both accompanists have 3 hours of free time, which gives more space to schedule exams as preferred. If both accompanists had, say, 30 minutes of free time, I would not want to adjust this algorithm at all. This is because it is maximising the possibility of scheduling all examinees, and in this situation, it needs to be maximised.

To clarify, I would like to explore the possibility of ignoring this freeTimeSort function in certain situations, when the scheduling is not that tight. At the start of the schedule function, this must run, as it sorts everyone by free time and creates a good list of priority. However, in future iterations, I could tweak the freeTimeSort function to ignore sorting if free time is sufficiently large. I have updated the code as shown below.

```

function freeTimeSort($rowA, $rowB) {
    if ($rowA[3] > 90 and $rowB[3] > 90){
        return 0;
    } else {
        return $rowA[3] - $rowB[3];
    }
}

function freeTimeSort_initial($rowA,$rowB){
    return $rowA[3] - $rowB[3];
}

```

```

// sort ascending by free time
if ($time->format('H:i')=='09:00'){
    //sort strictly at the start
    usort($accompIDs,'freeTimeSort_initial');
} else {
    //sort with some leniency
    usort($accompIDs,'freeTimeSort');
}

```

If it is 09:00, the accompanists are sorted strictly (as usual), because this allows examineeIDs to be sorted by priority. As exams are scheduled, the priority may slightly

change, but the general order will usually be quite similar to the original one. Therefore, it is unnecessary to change the order of accompanists if they're not very urgent. I thought about quantifying the level of urgency by how different \$rowA[3] and \$rowB[3] are (by how much the 2 free times differ). However, this would still be quite lenient even when stricter scheduling is needed. Instead I decided to create a leniency threshold, a number which can decide whether to sort accompanists by priority, or leave them in their original order. I adjusted the function to return 0 if the free time of both accompanists is greater than 90 minutes. This number is just an estimate, but roughly determines its leniency. If reduced it becomes more lenient (and prefers grouping), and if increased it becomes more strict (and maximises efficiency).

This change means that, above a certain threshold, the accompanist order doesn't change. This makes sense, as there is no need to try and fit accompanists in as carefully as possible, when there is clearly enough free time for this schedule to work well. Below this threshold, free time is quite low, so the exams should be sorted to ensure efficient scheduling. Therefore, I revert to the original sorting algorithm. This should solve my issue, but it may need tweaking to become more accurate. I will run this test again to check.

Purpose of test	Test data	Expected Result	Successful?
Testing how the program responds to too many examinees being inputted	Using fake data in the database  Threshold of 90 minutes (see above)	End of day message and list of unscheduled examinees  Schedule displayed with line breaks whenever there is a break.	Almost (see below)

The day has ended.  
The following examinees have not been scheduled: Isabella Scott , Ava Moore , Liam White

## Exam Schedule

Student Name	Instrument	Grade	Accompanist Name	Start Time
Sophie Hill	Violin	1	Christopher Smith	09:00
Chloe Harrison	Jazz Saxophone	1	Christopher Smith	09:13
Carter Evans	Jazz Saxophone	5	Christopher Smith	09:26
Sophia Hall	Trombone	2	Sarah Johnson	09:44
Daniel Williams	Trombone	5	Sarah Johnson	09:57
Noah Turner	Trombone	5	Sarah Johnson	10:15
Olivia Jones	Trumpet	8	Sarah Johnson	10:48
Jane Smith	Violin	2	Sarah Johnson	11:16
Avery Cooper	Violin	5	Sarah Johnson	11:29
Emma Davis	French Horn	1	David Williams	11:47
Mia Baker	Trumpet	5	David Williams	12:00
Emily Miller	Classical Guitar	3	David Williams	12:18
Logan Wilson	Classical Guitar	3	David Williams	13:31
Joe Davies	Classical Guitar	4	David Williams	13:44
Alex Johnson	Classical Guitar	6	David Williams	14:02
Ethan Clark	Trumpet	1		14:25
Henry Reed	Clarinet	5	Sarah Johnson	14:38
Michael Brown	Trumpet	7		14:56
Madison Barnes	Classical Guitar	1		15:34
Aiden Fisher	Classical Guitar	3		15:47
Lucas Ross	Viola	3	David Williams	16:00
Lily Morris	Clarinet	1	David Williams	16:13
Jackson Ward	Classical Guitar	4		16:26
Grace Kelly	Jazz Saxophone	3		16:44

The first improvement to make from the previous test is now complete. In the alert message, the names of unscheduled examinees are now displayed. I believe that this makes the message more user-friendly and abstracts all unnecessary information, only displaying what is actually needed.

The second improvement has got some interesting results. The remedial action I made has seemed to be more lenient when sorting by free time, which has fixed this issue of alternating accompanists! The first session now has 2 accompanist blocks, one for Christopher Smith and one for Sarah Johnson. This is a good sign, as the code is now preferring to stay in groups instead of strictly sorting by free time, as long as the free time is large enough. It tends to stay in the groups that were originally set for them by 'freeTimeSort\_initial'. However, it seems that the threshold may be too low, as Sarah Johnson is being preferred over David Williams for longer than expected. Grouping

accompanists is great, but too large groups can tire accompanists out. Ms Pearcey told me this during the start of my project, so I am wary not to simply schedule all accompanists in the order calculated by freeTimeSort\_initial. Although not clearly shown here, if the accompanist availability was stricter then it is possible that the threshold is so low that it would return an error, asking for more availability. I will try to avoid this as much as I can, as I would definitely not want to waste the accompanist's time. The key is to find a middle ground between grouping accompanists together and optimising their time. My original method optimised their time too much, resulting in the alternating pattern. This is grouping too much, which leads to the accompanist's time being wasted. I will try again with a threshold of 150 minutes, as I want to give more bias towards optimising the accompanist's time. I would rather see slightly poor grouping, and better use of their time, than what is being displayed currently.

```
if ($rowA[3] > 150 and $rowB[3] > 150){
    return 0;
```

Purpose of test	Test data	Expected Result	Successful?
Testing how the program responds to too many examinees being inputted	Using fake data in the database Threshold of 150 minutes (see above)	End of day message and list of unscheduled examinees  Schedule displayed with line breaks whenever there is a break.	Yes

## Exam Schedule

Student Name	Instrument	Grade	Accompanist Name	Start Time
Sophie Hill	Violin	1	Christopher Smith	09:00
Chloe Harrison	Jazz Saxophone	1	Christopher Smith	09:13
Carter Evans	Jazz Saxophone	5	Christopher Smith	09:26
Sophia Hall	Trombone	2	Sarah Johnson	09:44
Daniel Williams	Trombone	5	Sarah Johnson	09:57
Noah Turner	Trombone	5	Sarah Johnson	10:15
Olivia Jones	Trumpet	8	Sarah Johnson	10:48
Emma Davis	French Horn	1	David Williams	11:16
Mia Baker	Trumpet	5	David Williams	11:29
Emily Miller	Classical Guitar	3	David Williams	11:47
Logan Wilson	Classical Guitar	3	David Williams	12:00
Joe Davies	Classical Guitar	4	David Williams	12:13

Alex Johnson	Classical Guitar	6	David Williams	13:31
Lucas Ross	Viola	3	David Williams	13:54
Lily Morris	Clarinet	1	David Williams	14:07
Ethan Clark	Trumpet	1		14:20
Jane Smith	Violin	2	Sarah Johnson	14:33
Avery Cooper	Violin	5	Sarah Johnson	14:46
Henry Reed	Clarinet	5	Sarah Johnson	15:04
Michael Brown	Trumpet	7		15:37
Madison Barnes	Classical Guitar	1		16:00
Aiden Fisher	Classical Guitar	3		16:13
Jackson Ward	Classical Guitar	4		16:26
Grace Kelly	Jazz Saxophone	3		16:44

I believe that I have found an optimal value for the leniency threshold. The alternating pattern has been removed, and accompanists are grouped appropriately into good sizes. Currently, the value of this threshold is 150 minutes. If I encounter similar issues later on, I may adjust this value accordingly. As each schedule can be so different from each other, the ideal value of this threshold may differ with each input. However, any further improvements to this feature would be out of the scope of this project. I am now happy with the entire scheduling process, and the way in which it determines what exam to schedule next. By introducing some leniency into the system, my problem of alternating accompanists has been solved.

Purpose of test	Test data	Expected Result	Successful?
Testing that the schedule function recognises insufficient accompanist availability	Using fake data in the database	<p>Alert message detailing which examinees are unscheduled.</p> <p>Error message detailing which accompanists need to be adjusted.</p> <p>Schedule displayed up until the point where the error is found.</p>	Yes

The following examinees have not been scheduled: Chloe Harrison , Carter Evans , Lily Morris , Jane Smith , Avery Cooper , Henry Reed , Ethan Clark , Michael Brown , Madison Barnes , Aiden Fisher , Jackson Ward , Isabella Scott , Ava Moore , Liam White , Grace Kelly

ERROR:

Christopher Smith has too many exams / is not available enough. They need at least 7 minutes more.  
David Williams has too many exams / is not available enough. They need at least 6 minutes more.

### Exam Schedule

Student Name	Instrument	Grade	Accompanist Name	Start Time
Emma Davis	French Horn	1	David Williams	09:00
Mia Baker	Trumpet	5	David Williams	09:13
Emily Miller	Classical Guitar	3	David Williams	09:31
Logan Wilson	Classical Guitar	3	David Williams	09:44
Joe Davies	Classical Guitar	4	David Williams	09:57
Sophia Hall	Trombone	2	Sarah Johnson	10:15
Daniel Williams	Trombone	5	Sarah Johnson	10:28
Alex Johnson	Classical Guitar	6	David Williams	11:01
Noah Turner	Trombone	5	Sarah Johnson	11:24
Lucas Ross	Viola	3	David Williams	11:42
Olivia Jones	Trumpet	8	Sarah Johnson	11:55
Sophie Hill	Violin	1	Christopher Smith	12:23

This test shows that the feature of checking accompanist availability is successful. I reduced the availability in the ExamAccomps table for Christopher Smith and David Williams, so this message has been displayed as expected. All unscheduled examinees have been correctly displayed as well. The only issue with this feature is that the value of 7 and 6 minutes more in the error message is not very accurate. It is very likely that the actual number is around 30-45 minutes each. I encountered this issue during unit testing, so decided to emphasise the words 'at least' by underlining it. This conveys the message more clearly. I could try to calculate a more accurate value for this number, but those calculations would become quite complex. As accompanist availability overlaps, but only one exam can be scheduled at a time, I would have to consider everyone's availability in these calculations. This is not within the scope of my project, so I will not make any more adjustments to this feature.

The full schedule function is shown below.

```

406 function schedule($conn){
407     $emptySchedule="TRUNCATE TABLE Schedule";
408     $conn->query($emptySchedule);
409     // initialise variables/classes/constants
410     $numCompletedBreaks=0;
411     $time = new DateTime('09:00');
412     $sessionTime=0;
413     $dayExamTime=0;
414     $eod=false;
415     $error="";
416     $interval = new DateInterval("PT10M");
417
418     //create examineeIDs
419     $examineeSQL="SELECT ExamineeID FROM Examinees";
420     $examineeResult=$conn->query($examineeSQL);
421     $examineeIDs = [];
422     while ($row = $examineeResult->fetch_assoc()) {
423         $examineeIDs[] = $row['ExamineeID'];
424     }
425     while (count($examineeIDs)!=0 and $eod==false and $error==""){
426         //sort examinees by priority and other factors
427         [$examineeIDs,$miniError]=recalculateOrder($examineeIDs,$time,$conn);
428         $error.= $miniError; //append miniError to error
429
430         //search for available accompanist
431         $available=false;
432         $i=0;
433         while ($available == false and $i<count($examineeIDs)){
434             $available = accompAvailable($examineeIDs[$i],$time,$conn);
435             if (!$available){
436                 //iterate through examineeIDs
437                 $i++;
438             }
439         }
440     }

```

```

441     if($available){
442         $ExamineeID=$examineeIDs[$i];
443
444         //get duration of examinee
445         [$duration,$accompID] = getExamineeInfo($ExamineeID,$conn);
446
447         //validate the possible schedule change
448         $newTime= clone $time;
449         $newTime->modify("+{$duration} minutes");
450         $newSessionTime = $sessionTime + $duration;
451         $newDayExamTime = $dayExamTime + $duration;
452         $isValid = meetingRegulations($newTime,$newDayExamTime,
453         $newSessionTime,$numCompletedBreaks);
454
455         [$nextBreak,$miniError]=nextBreak($numCompletedBreaks);
456         $error.= $miniError;
457         $upperBound = $nextBreak->add($interval);
458
459         if ($newTime<=$upperBound and $isValid){
460
461             // insert into schedule
462             $StartTime=$time->format('H:i');
463             $schedulingSQL="INSERT INTO Schedule (StartTime,ExamineeID)
464             | VALUES ('$StartTime','$ExamineeID')";
465             $conn->query($schedulingSQL);
466
467             // remove examinee from examineeIDs
468             unset($examineeIDs[$i]);
469             $examineeIDs=array_values($examineeIDs);
470
471             // update time variables
472             $time=$newTime;
473             $sessionTime=$newSessionTime;
474             $dayExamTime=$newDayExamTime;
475
476         } else {
477             [$time,$sessionTime,$numCompletedBreaks,$eod,$miniError]
478             =addBreak($time,$numCompletedBreaks);
479             $error.= $miniError;
480         }

```

```

481     } else {
482         //find bounds for ideal break time
483         [$nextBreak,$miniError]=nextBreak($numCompletedBreaks);
484         $error.= $miniError;
485         $upperBound = $nextBreak->add($interval);
486         $lowerBound = $nextBreak->sub($interval);
487
488         //can a break be scheduled now?
489         if ($time >= $lowerBound and $time <= $upperBound){
490             [$time,$sessionTime,$numCompletedBreaks,$eod,$miniError]
491             =addBreak($time,$numCompletedBreaks);
492             $error.= $miniError;
493         } else {
494             //find the time difference until the next 30 min slot
495             [$difference,$currentField]=timeDifference($time);
496
497             //validate the possible schedule change
498             $newTime= clone $time;
499             $newTime->modify("+$difference minutes");
500             $newSessionTime = $sessionTime + $difference;
501             $isValid = meetingRegulations($newTime,$dayExamTime,
502             $newSessionTime,$numCompletedBreaks);
503
504             if ($isValid){
505                 //if valid, increment time to next 30min slot
506                 $time=$newTime;
507                 $sessionTime=$newSessionTime;
508             } else {
509                 //if invalid, schedule a break
510                 [$time,$sessionTime,$numCompletedBreaks,$eod,$miniError]
511                 =addBreak($time,$numCompletedBreaks);
512                 $error.= $miniError;
513             }
514         }
515     }
516 }
517

```

```

517
518     $alert="";
519     if ($eod){
520         //display end of day message
521         $alert.= "The day has ended.<br> ";
522     }
523     if (count($examineeIDs) != 0){
524         // display unscheduled examinee names
525         $names=[];
526         for ($i=0 ; $i<count($examineeIDs) ; $i++) {
527             // get examinee name
528             $getNames = "SELECT FirstName, LastName
529             FROM Students WHERE StudentID='".$examineeIDs[$i]'";
530             $nameResult = $conn->query($getNames);
531             $nameRow=$nameResult->fetch_assoc();
532             $fname=$nameRow['FirstName'];
533             $lname=$nameRow['LastName'];
534             //add name to array
535             $names[]=$fname $lname";
536         }
537         //add all names to alert message
538         $alert.="The following examinees have not been scheduled: "
539         .implode(" , ",$names)."<br>";
540     } else {
541         $alert.="All examinees have been scheduled!";
542     }
543
544     return [$alert,$error];
545 }
546

```

### 3.9.3 Review of stage 9

#### *Stage 9 summary*

- Developed the schedule function to create a full schedule from a valid database
- Combined functions from stages 4 to 8 in order to create this feature
- Created an easy-to-use interface for scheduling exams in ‘schedule.php’
- Displayed the full schedule in ‘schedule.complete.php’

#### *Testing carried out*

- As several modules from previous stages have been combined in this stage, I conducted a full system test of the scheduling functionality
- Tested that the function creates a valid schedule
- Tested that the system adheres to accompanist availability and groups exams correctly
- Tested that the schedule is displayed in the most helpful way, abstracting all irrelevant information
- Tested that the schedule responds well to extended unavailability from accompanists

- Tested that the system calculates which accompanists have submitted insufficient availability
- Tested that the system creates a valid schedule when too many examinees are submitted
- Fixed the ‘alternating accompanists’ problem by introducing a leniency threshold
- Tested that all error and alert messages are informative and direct

All tests were successful in the end.

### *Success criteria met*

I can confidently say that success criteria S1-8 have been met now. My system testing has shown that the produced schedule meets TCL requirements through constant validation from the meetingRegulations function, meeting S1. The process of sorting accompanists by free time means that the most urgent exams are scheduled first, so minimal time is wasted and the number of examinees scheduled is maximised. This meets S2 and S3. The function recalculateOrder will allow each accompanist’s exams to be grouped. The new addition of this leniency threshold allows me to find a balance between grouping accompanists and scheduling the most urgent exams. I believe that I have found this balance, so S4 has been met.

The function recalculateOrder (more specifically the instrumentSort function) allows exams to be grouped into instrument families, then instruments, and sorted by grade, meeting S5. Integrating the accompAvailable function into the system allows the schedule to always adhere to accompanist availability, so S6 is met. recalculateOrder checks if free time is negative for any accompanists, allowing the system to determine if an accompanist has submitted insufficient availability and display an error message accordingly. This meets S7. Finally, the schedule function itself will insert the start time and ExamineeID into the ‘Schedule’ table in the database. This database stores the schedule securely, allowing it to be accessible at any time, so S8 has been achieved.

### *Changes to the design*

There have been some notable changes to the design, but the fundamental aspects have stayed the same. I have found that the detailed design conducted in section 2.5.2 has greatly streamlined the development of these modules.

I have included more validation in the system, and many functions will return error messages (not just recalculateOrder as designed). This makes my system more robust. These error messages are all combined into \$error. \$alert is a new addition, and will display some more helpful information. It will output any unscheduled examinees, or say that all examinees have been scheduled. It will also mention if the end of the day has been reached. This change makes my system more informative and easier to use. It gives Ms Pearcey all of the information needed to make the necessary changes to the data.

To improve modularity in the system, the function timeDifference has been created. This ensures consistency and simplifies overall maintenance of the product. I have also created a much more informative and usable interface to display the schedule. Alert and error messages are shown in highlighted boxes, using colours to emphasise their role. I have

calculated when a break has been scheduled and displayed an empty row, to split up the table by session.

To ensure validity of the schedule in any case, I have changed the process of incrementing the time (if no accompanists are available). `meetingRegulations` validates the proposed change before it is done, and tries to schedule a break if possible. This gives me confidence that S1 is met, no matter the input.

I introduced a leniency threshold when sorting accompanists by free time. If accompanist free time is greater than 150 minutes, there is no need to perfectly order each and every accompanist by free time. They are already ordered by free time at the start, so as the schedule is created the priority may change slightly. This threshold will only change the order of accompanists if their free time is below 150 minutes. This means that the function prefers to group accompanists rather than be as efficient as possible **if free time is over 150 minutes**. This is because the accompanists have so much free time that there is a very low chance of them running out of free time. The function will try to be as efficient as possible below this threshold as the accompanists do not have much free time remaining, so efficiency is maximised.

### *Review of the project so far*

The success of this function means that the most complex and important aspect of this project is complete. Ms Pearcey will be able to use this system to easily create schedules, greatly reducing her workload.

I have not developed the squeeze function yet. This function is quite complex, as its algorithm in section 2.5.2 shows, and is only used in a limited range of situations. This function will try to make sure that the breaks start very close to their ideal time, and try to maximise the number of examinees scheduled. After testing the system in this stage, I am confident that the maximum number of examinees are being scheduled without this function, and break times are still very close to their ideal times. I do not see the need to develop such a complex function if it will have minimal impact on the solution, given the time constraints of this project. Therefore, it will not be developed.

SoundBro currently has 2 major modules: the login/registration module, and the scheduling module. These are separate from each other, so need to be combined together in the next section to create a full prototype of my solution. I will now move on to using real data provided by Ms Pearcey. This allows me to assess how the system responds to real input data, so I can gain some insight into what should be changed.

I am very pleased with the state of the database. It has been developed exactly as designed, and executing the correct SQL queries allows any combination of data to be obtained. I have struggled a lot with making the right SQL queries, but I am impressed with the consistency of its accuracy. As the database is in third normal form, accessing and maintaining the database is simplified. I have felt the benefits of my thorough design in recent stages of development, as more and more SQL queries have been made, but all have consistently given the correct results.

The directory structure has changed as I have created ‘schedule-complete.php’ to display the interface.

```
soundbro/
|--- index.php
|--- login.php
|--- register.php
|--- register-student.php
|--- register-supervisor.php
|--- dashboard.php
|--- schedule.php
|--- schedule-complete.php
|--- logout.php
|--- styles.css
|--- includes/
    |--- db-connect.php
    |--- navbar.php
    |--- footer.html
|--- logo/
    |--- full.png
    |--- favicon3.png
```

## 3.10 System integration

### 3.10.1 Developing and testing

I currently have 2 large modules: one for login functionality, and one for scheduling functionality. These both use a consistent interface, but are not integrated properly with each other. In the interest of developing a modular solution, I avoided combining the 2 modules until both have been rigorously tested. Testing at this lower level allows me to carefully understand each error and make the system fundamentally more robust and accurate. Now that both have undergone system testing and all errors have been removed, I am ready to integrate them properly into one product.

At the end of this stage, I will use real data provided by Ms Pearcey to conduct a larger system test. In order to test the system effectively, it is best to assess how it performs when given real life scenarios. The fake data allowed me to manipulate any inputs to comprehensively test each and every part of the system, through unit testing and the above system test. Now, real data will test the overall performance of the system. Feedback from Ms Pearcey will clarify which aspects of the system are working well and which need improvement. Her perspective is crucial to this project, as I aim to reduce her workload.

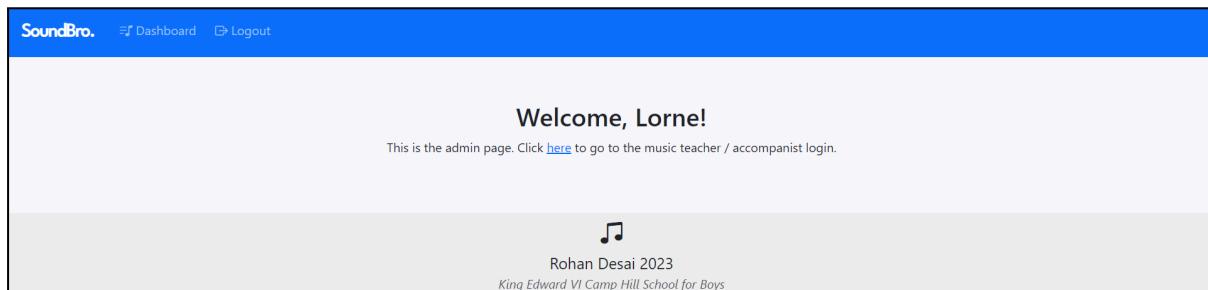
There are few key things to keep in mind when integrating these 2 modules together:

- I have not yet built the feature of inputting data into the database. This will be added in a later stage. As a result, my prototype will skip over some parts of the scheduling process. My swimlane diagram in section [2.2.2](#) shows the order of events when scheduling exams, and gathering inputs is clearly between these 2 modules. For now, some parts of this prototype will essentially assume that data is already in the database. Later iterations will improve upon this.
- There are 3 groups of users: students, supervisors, and the admin. Only the admin should be able to access scheduling functionality.

Once Ms Pearcey logs into the system, she is taken to the dashboard (shown below).

The screenshot shows a web application interface titled "SoundBro". At the top, there is a blue header bar with the "SoundBro." logo, a "Dashboard" link, and a "Logout" link. Below the header, the word "Dashboard" is centered in a large, bold, black font. Underneath it, the text "Welcome, Lorne!" is displayed. A message box contains the text: "Welcome! If you are here for admin purposes, click [here](#). If not, stay on this page to see music teacher / accompanist info." At the bottom of the screen, there is a light gray footer bar featuring a small musical note icon, the text "Rohan Desai 2023", and "King Edward VI Camp Hill School for Boys".

This page will display any music teacher / accompanist related information. This is because Ms Pearcey is both an admin and a supervisor. This page is for her role as a supervisor, but it links to the admin dashboard, shown below. This page is called 'dashboard-admin.php'.



This dashboard should provide the link to the schedule functionality. I could simply copy all of the code from schedule.php into this page, but then this wouldn't be a dashboard. Doing this would make sense in the short term, but future iterations of this solution will require this page to simply display information about the schedule and link to many separate features. Therefore, copying and pasting schedule.php here would be a 'quick fix', but not sensible in the long term development of this project.

The dashboard should allow Ms Pearcey to do 2 things: view an existing schedule, or start a new schedule. Once the gathering inputs functionality is built, this can be directly linked to these 2 buttons. When a new schedule is started, the following tables will be emptied using the SQL TRUNCATE command: BasicInfo, Examinees, ExamAccomps, and Schedule. As tables are emptied instead of deleted, the code differs from what I have originally designed in [2.5.4](#), but it achieves the same goal. It should then take the user to a page where they can enter all of the inputs. This page will be schedule.php, and the 'schedule' button will act as a submit button for these inputs.

If a schedule has not been created, the button to view an existing schedule should not be displayed. If it has, then the user should be able to click this button, taking them to schedule-complete.php. This should integrate the 2 modules together seamlessly, creating an intuitive user experience.

## Welcome, Lorne!

This is the admin page. Click [here](#) to go to the music teacher / accompanist login.

### Schedule Music Exams

[View Existing Schedule](#)

[Start New Schedule](#)

**Warning:** Starting a new schedule will erase ALL information about the current exam cycle.

I developed this container in dashboard-admin.php. This clearly shows 2 buttons to represent the 2 possible actions Ms Pearcey can take. I thought about making these into 2 links, as viewing the schedule will just redirect the user. However, starting a new schedule will cause many tables in the database to be emptied, which a simple link using the `<a>` tag cannot do. I could just make the ‘view existing schedule’ button into an `<a>` tag, but this would make the interface look inconsistent. ‘Start New Schedule’ is displayed in red to convey a sense of danger, and the warning message below it should prevent misclicks. Informing the user with this message is simple but effective, and this on-screen help is something my usability features design (section [2.3](#)) has aimed to do.

The top button should only be displayed if the schedule exists. To check this, I have queried the database to find the number of rows in the Schedule table. If greater than 0, `scheduleExists` is set to true.

```
$scheduleExists=False;
// check status of scheduling process
$sql="SELECT * FROM Schedule";
$scheduleResult=$conn->query($sql);

// has schedule been made?
if ($scheduleResult -> num_rows > 0){
    $scheduleExists=True;
}
```

This variable can be used to determine whether or not to display the button.

```

<?php if ($scheduleExists) { ?>
<div class="m-2 p-2 form-group text-center">
|   <button type="submit" name="viewExisting" class="fs-4 btn btn-primary w-75 btn-block">View Existing Schedule</button>
</div>
</?php } ?>

<!-- start new schedule -->
<div class="m-2 p-2 form-group text-center">
|   <button type="submit" name="new" class="fs-4 btn btn-danger w-75 btn-block">Start New Schedule</button>
</div>
<?php if($scheduleExists){ ?>
    <p><strong>Warning: </strong>Starting a new schedule will erase ALL information about the current exam cycle.</p>
</?php } ?>

```

If true, ‘view existing schedule’ is displayed. Additionally, the warning message is only displayed if this variable is true. This is because there shouldn’t be a warning message if there is only one option (one button to click). If nothing has been scheduled, then nothing is in the database for this exam cycle. There is no risk if empty tables are being emptied, so no need for a message. If displayed, this could confuse the user as they would be unsure whether to click or not.

```

if (isset($_POST['new'])) {
    //empty all tables
    $clearBasicInfo="TRUNCATE TABLE BasicInfo";
    $clearExamAccomps="TRUNCATE TABLE ExamAccomps";
    $clearExaminees="TRUNCATE TABLE Examinees";
    $clearSchedule="TRUNCATE TABLE Schedule";

    $conn->query($clearBasicInfo);
    $conn->query($clearExamAccomps);
    $conn->query($clearExaminees);
    $conn->query($clearSchedule);

    header("Location: schedule.php");
    exit();
}

```

If a new schedule is started, the above commands will empty the relevant tables in the database. This prevents existing records in these tables from interfering with the inputted data. The user is then redirected to schedule.php. They will be able to enter data in this page, and then click ‘schedule’ to run the schedule function. The process of entering data and validation will be built in the next stage. The button to view an existing schedule simply redirects to schedule-complete.php

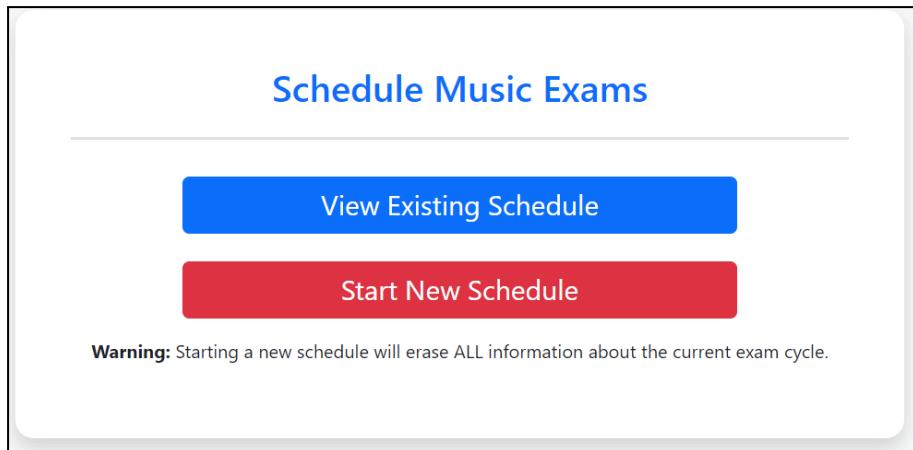
```

if(isset($_POST['viewExisting'])) {
    header("Location: schedule-complete.php");
    exit();
}

```

Now, the login functionality should be integrated into the schedule feature. I will run some tests to check that everything is working as expected.

Purpose of test	Test data	Expected Result	Successful?
'View Existing Schedule' button is displayed if the schedule exists	The table 'Schedule' contains the created schedule.	Button displayed: 'View Existing Schedule'  Message displayed: 'Warning: Starting a new schedule will erase ALL information about the current exam cycle.'	Yes



Purpose of test	Test data	Expected Result	Successful?
'View Existing Schedule' button takes user to schedule-complete.php	The table 'Schedule' contains the created schedule.	Redirected to schedule-complete.php, where the schedule is displayed.	Yes

All examinees have been scheduled!

Exam Schedule				
Student Name	Instrument	Grade	Accompanist Name	Start Time
Emma Davis	French Horn	1	David Williams	09:00
Mia Baker	Trumpet	5	David Williams	09:13
Emily Miller	Classical Guitar	3	David Williams	09:31
Logan Wilson	Classical Guitar	3	David Williams	09:44
Joe Davies	Classical Guitar	4	David Williams	09:57
Sophia Hall	Trombone	2	Sarah Johnson	10:15
Daniel Williams	Trombone	5	Sarah Johnson	10:28
Alex Johnson	Classical Guitar	6	David Williams	11:01

Purpose of test	Test data	Expected Result	Successful?
'Start New Schedule' empties the relevant table in the database	Database is populated with data	Redirected to schedule.php Following tables are emptied: - BasicInfo - Examinees - ExamAccomps - Schedule	Yes

The image contains four vertically stacked screenshots from MySQL Workbench. Each screenshot shows a database table with a green message box at the bottom stating 'MySQL returned an empty result set (i.e. zero rows). (Query took 0.00XX seconds.)'. The tables shown are examaccomp, basicinfo, examinees, and schedule.

- Table: examaccomp**: Shows a green message box: 'MySQL returned an empty result set (i.e. zero rows). (Query took 0.0028 seconds.)'
- Table: basicinfo**: Shows a green message box: 'MySQL returned an empty result set (i.e. zero rows). (Query took 0.0011 seconds.)'
- Table: examinees**: Shows a green message box: 'MySQL returned an empty result set (i.e. zero rows). (Query took 0.0053 seconds.)'
- Table: schedule**: Shows a green message box: 'MySQL returned an empty result set (i.e. zero rows). (Query took 0.0023 seconds.)'

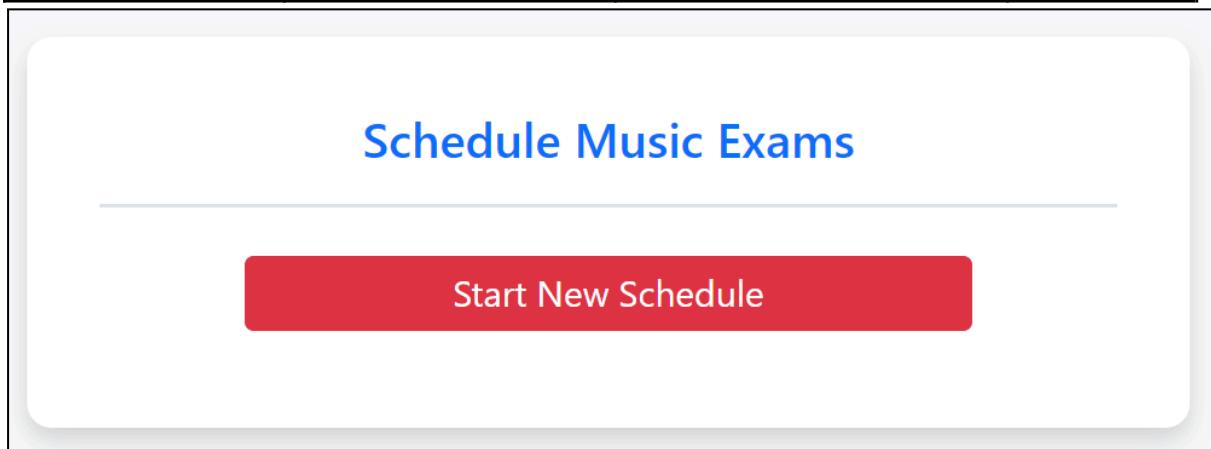
The top right of each screenshot shows the table name, and all are empty (as expected).

The image shows a screenshot of a web browser displaying the URL 'localhost/soundbro/schedule.php'. The page has a blue header bar with the SoundBro logo and navigation links for Dashboard and Logout. Below the header is a large white box containing a heading 'Schedule Music Exams' and a message 'Click the button below to schedule your TCL music exam!'. At the bottom of the box is a large blue button with the text 'Schedule'.

To prevent actually losing all of my data, I duplicated the database and all of its data. This test was performed on a copy of the database. At this point, the data would manually be

entered into the database before clicking ‘schedule’. The next stage will build the feature of entering data on this page.

Purpose of test	Test data	Expected Result	Successful?
‘View Existing Schedule’ is not displayed if schedule doesn’t exist	The table ‘Schedule’ is empty.	‘Start New Schedule’ is displayed. ‘View Existing Schedule’ is not displayed. The warning message is not displayed	Yes



All changes to the code in this stage have now been tested. I am ready to conduct a system test using real data, and hope to obtain stakeholder feedback from Ms Pearcey.

### 3.10.2 System test with real data

In order to test the system effectively, it is best to assess how it performs when given real life scenarios. The fake data allowed me to manipulate any inputs to comprehensively test each and every part of the system, through unit testing and the above system test. Now, real data will test the overall performance of the system, feedback from Ms Pearcey will highlight what needs improvement.

In order to effectively use this real data, I will copy the exact structure of the database into a new database, called ‘db\_winter\_23’. This prevents any fake data from interfering with the system. I will register each user using CSV files imported into phpMyAdmin. I will not use real email addresses, but I will use real names. For this system test, I will use the data provided for the **Winter 2023** cycle. This data is displayed below. Accompanist availability is shown at the bottom-right of the image.

Candidate's first name	Candidate's last name	Instrument	Grade	Teacher	Exam length	Accompanist
Aakshat	Kumar	Classical Guitar	Grade 4	Miss Palmer	00:18	-
Ahana Aditi	Seeam	Classical Guitar	Grade 1	Mr Phillips	00:13	-
Aisha	Nashmia	Classical Guitar	Grade 1	Mr Phillips	00:13	-
Akshay	Suglani	Saxophone	Grade 3	Mr Drew	00:13	ED
Aliza	Rayhan	Flute	Grade 6	Mr Hay	00:23	SFP
Ananya	Tare	Classical Guitar	Grade 4	Mr Phillips	00:18	-
Anisha	Kawle	Classical Guitar	Grade 3	Mr Phillips	00:13	-
Aroush	Haider	Classical Guitar	Grade 1	Mr Phillips	00:13	-
Daniel	Okpla	Classical Guitar	Grade 2	Mr Rose	00:13	-
Doyinsola	Oliyide	Saxophone	Grade 1	Mr Drew	00:13	ED
Eshal	Aamir	Classical Guitar	Grade 1	Mr Phillips	00:13	-
Farishta	Dosanjh	Classical Guitar	Grade 1	Mr Phillips	00:13	-
Henna	Naveed	Clarinet	Grade 5	Mr Meadows	00:18	JM
Jiayi (Eva)	Chen	Clarinet	Grade 3	Mr Meadows	00:13	JM
Lauren	Pumphrey	Flute	Grade 6	Mr Hay	00:23	SFP
Leya	Rahman	Trumpet	Grade 5	Mrs Butler	00:18	SFP
Lily-Marie	Le Blanc	Viola	Grade 3	Ms Keum	00:13	SFP
Maryam	Rahman	Violin	Grade 5	Mr McGee	00:18	SFP
Milaura	Fernando	Violin	Grade 1	Mr McGee	00:13	SFP
Nikhil	Gilliam	Classical Guitar	Grade 4	Miss Palmer	00:18	-
Shailey	Sivathasan	Classical Guitar	Grade 1	Mr Phillips	00:13	-
Sri	Grandhi	Classical Guitar	Grade 3	Mr Rose	00:13	-
Tula	Hobbs	Classical Guitar	Grade 5	Mr Phillips	00:18	-
Xander	Davis	Trumpet	Grade 6	Mrs Butler	00:23	SFP
Zaid	Rassam	Trumpet	Grade 1	Mrs Butler	00:13	SFP

In order to populate the Students table, I took the first and last names of each student and generated a fake email address for each: `firstname.lastname@example.com` . The real data allows me to compare mine and Ms Pearcey's schedules, but the email addresses do not have to be real for this system to work. For the parents, I did the same, using `lastname.parent@example.com` this time. For all users, I made the same password: `Password123`. However, I used `password_hash` to generate 25 hashes of this string. As the function uses an additional process called 'salting', each hash is different. The data created is shown below (the hashed password is longer than shown).

FirstName	LastName	Email	ParentEmail	Password
Aakshat	Kumar	aakshat.kumar@example.com	kumar.parent@example.com	\$2y\$10\$1osLt6/uz6906o
Ahana Aditi	Seeam	ahana.seeam@example.com	seeam.parent@example.com	\$2y\$10\$4WxikCExlq7IW
Aisha	Nashmia	aisha.nashmia@example.com	nashmia.parent@example.com	\$2y\$10\$9DBtlHYdnE5dF
Akshay	Suglani	akshay.suglani@example.com	suglani.parent@example.com	\$2y\$10\$dHZ.KW9aTkV1
Aliza	Rayhan	aliza.rayhan@example.com	rayhan.parent@example.com	\$2y\$10\$Ns/F3RepLogN8
Ananya	Tare	ananya.tare@example.com	tare.parent@example.com	\$2y\$10\$RnUOrrUdLoDG
Anisha	Kawle	anisha.kawle@example.com	kawle.parent@example.com	\$2y\$10\$RkBxDxREMXH9
Aroush	Haider	aroush.haider@example.com	haider.parent@example.com	\$2y\$10\$dtOKE6BY6Fjh2
Daniel	Okpla	daniel.okpla@example.com	okpla.parent@example.com	\$2y\$10\$u5IPkYf8aOw.xz
Doyinsola	Oliyide	doyinsola.oliyide@example.com	oliyide.parent@example.com	\$2y\$10\$GaO9GWepMEY
Eshal	Aamir	eshal.aamir@example.com	aamir.parent@example.com	\$2y\$10\$e/wQcDSrl85iGv
Farishta	Dosanjh	farishta.dosanjh@example.com	dosanjh.parent@example.com	\$2y\$10\$lhdhQantZsk7n1
Henna	Naveed	henna.naveed@example.com	naveed.parent@example.com	\$2y\$10\$ELORCp8YBlm
Jiayi (Eva)	Chen	jiayi.chen@example.com	chen.parent@example.com	\$2y\$10\$nFBUw4O8Q.Kc
Lauren	Pumphrey	lauren.pumphrey@example.com	pumphrey.parent@example.com	\$2y\$10\$j3sD/BdnwhyWE
Leya	Rahman	leya.rahman@example.com	rahman.parent@example.com	\$2y\$10\$KGSxEr/zWIAy
Lily-Marie	Le Blanc	lily-marie.le@example.com	leblanc.parent@example.com	\$2y\$10\$M.s3VCYsROpN
Maryam	Rahman	maryam.rahman@example.com	rahman2.parent@example.com	\$2y\$10\$cFt7xNY6oy5Oa
Milaura	Fernando	milaura.fernando@example.com	fernando.parent@example.com	\$2y\$10\$ScOUCU4yNgsz
Nikhil	Gilliam	nikhil.gilliam@example.com	gilliam.parent@example.com	\$2y\$10\$Xllp65/P4pTIO5
Shailey	Sivathasan	shailey.sivathasan@example.com	sivathasan.parent@example.com	\$2y\$10\$Ngx/vavLGj17Yj
Sri	Grandhi	sri.grandhi@example.com	grandhi.parent@example.com	\$2y\$10\$88kZER.OLZxE8
Tula	Hobbs	tula.hobbs@example.com	hobbs.parent@example.com	\$2y\$10\$jrfGWf5w0xN36
Xander	Davis	xander.davis@example.com	davis.parent@example.com	\$2y\$10\$ZGIEFGNG58O
Zaid	Rassam	zaid.rassam@example.com	rassam.parent@example.com	\$2y\$10\$4DwLxU/6fcRkd0

I converted this into a CSV file, and imported it into the database. I did the same for importing supervisors into the Supervisors table. I also manually registered Lorne Pearcey with admin@email.com (Ms Pearcey). This email is recognised by the system as the admin email, which gives her access to all scheduling features. When inputting accompanist availability into ExamAccomps, I manually entered the appropriate data into the database. The completed table ExamAccomps is shown below, along with the relevant entries in Supervisors.

AccompID	School	Start0900	Start0930	Start1000	Start1030	Start1100	Start1130	Start1200	Start1230	Start1300	Start1330	Start1400	Start1430	Start1500	Start1530	Start1600	Start1630
10	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
3	0	1	1	1	1	1	0	0	0	0	0	0	0	0	0	0	0
6	0	0	1	1	1	1	1	1	1	1	1	1	1	1	0	0	0

SupervisorID	FirstName	LastName	Email	Password
3	Elliot	Drew	elliot.drew@example.com	\$2y\$10\$u5IPkYf8aOw.xzmYgmPskePsomHK.2EUuNSe.YU.KJi...
6	John	Meadows	john.meadows@example.com	\$2y\$10\$lhdhQantZsk7n1EfC7eoOq9uHEDNROvLtTiaU/79fm...
10	Simon	Palmer	simon.palmer@example.com	\$2y\$10\$u5IPkYf8aOw.xzmYgmPskePsomHK.2EUuNSe.YU.KJi...

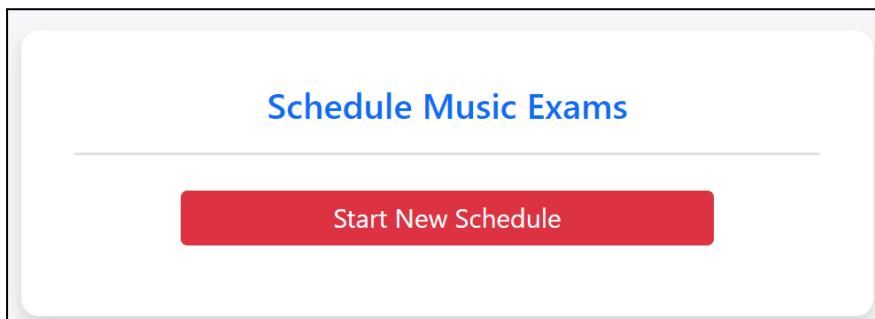
I created a record for BasicInfo manually. Then, I used the given data, as well as created IDs to manually create a CSV to populate the Examinees table. By replacing each teacher with

their SupervisorID, each accompanist with their AccomplID, and each examinee with their StudentID, I populated the Examinees table. The tables Families and Timings contain static data, so I exported from the original database and imported into the new one. Now, the database should be ready for the schedule function to run, I just need to connect my program to this new database. One benefit of the modularity of this system can be seen here. The file ‘db-connect.php’ stores the code that sets up the connection to the database, and this file is referenced on every page in my system. This makes it very easy to change the database, as changing one line of code will affect the entire product. This is shown below (line 6).

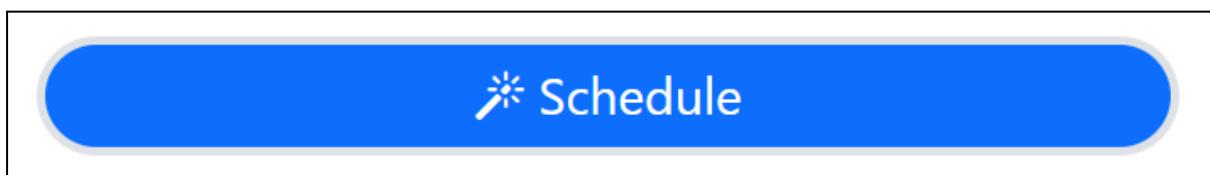
```
C: > wamp64 > www > soundbro > includes > db-connect.php > ...
1  <?php
2  // connect to database
3  $serverName = "localhost";
4  $serverUsername = "root";
5  $serverPassword = "";
6  $dbname = "soundBroDB2";
7  $conn = new mysqli($serverName, $serverUsername, $serverPassword, $dbname);
```

Finally, I registered ‘admin@email.com’ as a supervisor, which is recognised by the system as an admin login. The database should now be ready for the system test. My prototype of the solution has now combined several separate modules into a single, larger system. Using real data, I will test the functionality of this system and get feedback from Ms Pearcey. Her feedback will clarify the direction of this project’s development. This test will involve running through the entire process of scheduling exams in the system.

I logged in as an admin, taking me to the dashboard. From there, I clicked the link to get to the admin dashboard. At this point, the screen displayed one button, as shown below.



Clicking this button emptied several tables in the database. After clicking this, I populated the database as shown above. Now the database contains all valid data needed to produce a schedule. This button also redirected me to schedule.php, displaying the ‘schedule’ button.



Upon clicking this button, the schedule function ran to produce a schedule for the Winter 2023 exam cycle. This schedule is shown below.

All examinees have been scheduled!				
<b>Exam Schedule</b>				
<b>Student Name</b>	<b>Instrument</b>	<b>Grade</b>	<b>Accompanist Name</b>	<b>Start Time</b>
Doyinsola Oliyide	Saxophone	1	Elliot Drew	<b>09:00</b>
Akshay Suglani	Saxophone	3	Elliot Drew	<b>09:13</b>
Zaid Rassam	Trumpet	1	Simon Palmer	<b>09:26</b>
Jiayi (Eva) Chen	Clarinet	3	John Meadows	<b>09:39</b>
Henna Naveed	Clarinet	5	John Meadows	<b>09:52</b>
Leya Rahman	Trumpet	5	Simon Palmer	<b>10:10</b>
Xander Davis	Trumpet	6	Simon Palmer	<b>10:28</b>
Lily-Marie Le Blanc	Viola	3	Simon Palmer	<b>11:06</b>
Milaura Fernando	Violin	1	Simon Palmer	<b>11:19</b>
Maryam Rahman	Violin	5	Simon Palmer	<b>11:32</b>
Aliza Rayhan	Flute	6	Simon Palmer	<b>11:50</b>
Lauren Pumphrey	Flute	6	Simon Palmer	<b>12:13</b>
Ahana Seeam	Classical Guitar	1		<b>13:36</b>
Aisha Nashmia	Classical Guitar	1		<b>13:49</b>
Aroush Haider	Classical Guitar	1		<b>14:02</b>
Eshal Aamir	Classical Guitar	1		<b>14:15</b>
Farishta Dosanjh	Classical Guitar	1		<b>14:28</b>
Shailey Sivathasan	Classical Guitar	1		<b>14:41</b>
Daniel Okpla	Classical Guitar	2		<b>14:54</b>
Anisha Kawle	Classical Guitar	3		<b>15:07</b>
Sri Grandhi	Classical Guitar	3		<b>15:35</b>
Aakshat Kumar	Classical Guitar	4		<b>15:48</b>
Ananya Tare	Classical Guitar	4		<b>16:06</b>
Nikhil Gilliam	Classical Guitar	4		<b>16:24</b>
Tula Hobbs	Classical Guitar	5		<b>16:42</b>

This schedule has seemed to process real data very well! The schedule meets TCL regulations, schedules breaks at the correct times, groups accompanists well according to their availability, and groups exams correctly. Minimal time is wasted, and all examinees have been scheduled. All of the breaks start within 6 minutes of their ideal time, showing that the current algorithm works perfectly well at scheduling breaks. If the squeeze function was developed, it would have no impact on this schedule. This justifies its removal from the solution. After comparing this to all scheduling regulations mentioned in section 1.1.2/3, I cannot find any errors with this solution. The scheduling functionality is working as expected, and the entire prototype of the solution is working smoothly as one unit.

I showed the entire system and scheduling process to Ms Pearcey, so that she could provide feedback. She gave feedback in the form of the email shown below.

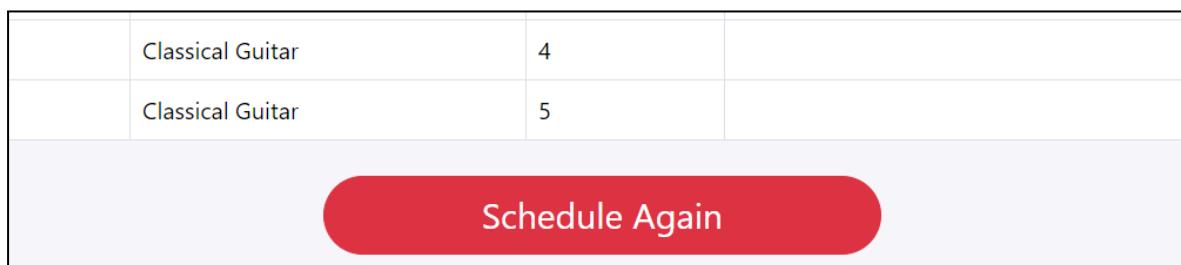
The screenshot shows an email inbox with one unread message. The subject of the email is "System test of CS project". The message is from Rohan Desai, sent at 07:45 (3 hours ago). Rohan writes: "Hi miss, Thanks for talking with me yesterday. I am conducting a system test of my project, so it would be great if you...". The message is then quoted by Lorne Pearcey, who replies at 09:31 (1 hour ago): "I think that the interface is very clean, clear and well-presented. All of the login screens and the actual timetable presentation are great! A pdf version to display or send to people would be good, but as it currently stands I could use a screenshot for this purpose as an interim measure". Lorne continues: "The actual scheduling does exactly what it should and uses the available time of the accompanists well. One thought for future development would be to consider putting an unaccompanied exam e.g. guitar, into a single slot, so that all accompanied exams using the same accompanist would be in a solid block; that is what I would probably aim to do myself. Although in this case, it would mean Mr Palmer having one exam after lunch, which is a worse scenario, so I would do what your program does! Alternatively, I would ask Mr Meadows if there was any chance of him coming in slightly earlier, but obviously this is not something that a program can do!". Lorne signs off with "Thank you!" and ends with three ellipsis dots.

This email shows that Ms Pearcey is very happy with the current prototype of the solution. She mentioned that the interface is very clean and clear, which shows that the usability features implemented have resulted in a positive user experience. The simplicity is intentional, so is the colour coordination and easy-to-press buttons. These were all designed in section [2.3](#), and now Ms Pearcey feels that the presentation is great as a result of these choices.

The actual scheduling was also praised by Ms Pearcey, she said it 'does exactly what it should' and efficiently uses accompanist availability. This is a great sign, showing that stages 4 to 9 do not need any further changes, as they have been developed correctly. She went on to suggest that the 09:26 exam could be replaced by an unaccompanied exam, which would prioritise grouping accompanists over using their time efficiently. However, she later changed her mind as this would put Mr Palmer in a worse position, forcing him to play for longer. She concluded by saying 'I would do what your program does!'. This is a very positive result, as Ms Pearcey has shown that she would originally schedule exams in a less effective way, but later change this to schedule it as I have. This shows that my program is working in a very effective way, and reaching these conclusions quicker than Ms Pearcey. This will lead to more efficient schedules being produced in shorter times, so the entire scheduling process can be streamlined. This gives me confidence to say that success criteria S1-8 have been met.

She said it would be good if I can make a PDF file of the schedule, as this would be sent to everyone and displayed on the notice board. This will be developed in future stages.

She also discussed how she would most likely ask Mr Meadows to come in earlier, so that he would be available for the 09:26 slot. My program cannot do this (as she mentioned), but it could support Ms Pearcey during this process. I can improve upon the schedule-complete.php functionality by creating a button that allows users to schedule again. This means that, after viewing the produced schedule, Ms Pearcey can talk to Mr Meadows and get him to adjust his availability. She can then go into the system, view the schedule, and click ‘schedule again’ directly from here, making any necessary changes. This button would work in the same way as the ‘Start new schedule’ button, emptying all databases and starting the process again. This promotes making changes to the schedule, as she can click ‘schedule again’ in just one click, without having to navigate back to the admin dashboard each time.



```

if (isset($_POST['new'])) {
  //empty all tables
  $clearBasicInfo="TRUNCATE TABLE BasicInfo";
  $clearExamAccomps="TRUNCATE TABLE ExamAccomps";
  $clearExaminees="TRUNCATE TABLE Examinees";
  $clearSchedule="TRUNCATE TABLE Schedule";

  $conn->query($clearBasicInfo);
  $conn->query($clearExamAccomps);
  $conn->query($clearExaminees);
  $conn->query($clearSchedule);

  header("Location: schedule.php");
  exit();
}
  
```

By using the same exact functionality as the ‘Start New Schedule’ button, I created this button. Both buttons also use the same colour to convey the same sense of danger, as this will cause many tables to be emptied. This button can be found at the bottom of the displayed schedule, which seems the most sensible place to put it. Once reading the entire schedule, naturally the page will be scrolled all the way down, so the button is placed here so that it can be pressed easily.

Ms Pearcey’s feedback has shown that this system is working very well, so this system test is complete.

### 3.10.3 Review of stage 10

#### *Stage 10 summary*

- Integrated the login functionality with the scheduling functionality
- Developed dashboard-admin.php
- Developed 'Start New Schedule' and 'View Existing Schedule' buttons

#### *Testing carried out*

- Carried out a few small tests of the buttons developed in this section, testing its functionality
- Carried out a large-scale system test on the entire prototype, using real-life data with feedback from my client, Ms Pearcey
- The system test and Ms Pearcey's feedback showed that the development of the solution so far is very successful, and there is not much (if anything) to improve on the current state of the prototype.

All tests were successful in the end.

#### *Success criteria met*

Ms Pearcey praised my interface in her response, so G1 is quite close to being met. Now that my entire solution is combined into one system, I am confident that criteria L1-4 have been met. Additionally, E1 has been met as the 'view existing schedule' button has been successfully built. This function will be improved in later stages as I make the displaySchedule function.

Success criteria E2 says that 'existing schedules are deleted at 22:00 on the final exam day'. However, in my design (section [2.5.1](#)), I removed this feature with an alternative that achieves the same purpose. The reason for this criteria is to make space for the next schedule, so that when a new schedule needs to be made the database is ready for new inputs. By designing and developing the 'Start New Schedule' button, the exact same goal is achieved, the only difference is the way in which it has achieved this goal. Therefore, I am happy to say that E2 has been met.

Success criteria S1-8 have been met, and Ms Pearcey's feedback has shown that it is doing 'exactly what it should' and making the same decisions as Ms Pearcey would make, in a shorter time frame. This reinforces the achievement of these criteria, and I will not develop these features any more.

#### *Changes to the design*

In this section, the 'Start New Schedule' button is slightly different than how it was originally designed. This is because tables are not being deleted and created again, but simply emptied using the TRUNCATE command. This minor change has no impact on the solution's functionality. Other changes include adding more on-screen help, and including a 'schedule again' button in 'schedule-complete.php'.

## *Review of the project so far*

I am pleased with the current state of this prototype. It has been quite successful, meeting all tests and functioning as expected. Ms Pearcey's feedback has shown that she feels the same way. I still need to build the functionality for gathering inputs, and publishing outputs. These will form the last 2 stages of my development.

My directory structure has been updated, as dashboard-admin.php was created to link the scheduling module to the login module. This addition is shown below. I also made changes to schedule-complete.php.

```
soundbro/
|--- index.php
|--- login.php
|--- register.php
|--- register-student.php
|--- register-supervisor.php
|--- dashboard.php
|--- dashboard-admin.php
|--- schedule.php
|--- schedule-complete.php
|--- logout.php
|--- styles.css
|--- includes/
    |--- db-connect.php
    |--- navbar.php
    |--- footer.html
|--- logo/
    |--- full.png
    |--- favicon3.png
```

## 3.11 Gathering inputs

### 3.11.1 Developing

To create a schedule, lots of data must be collected by the website. This data will populate the database, and section [2.4](#) and [2.5.4](#) details all data and validation needed for this process to work. The level of validation needed to make this system robust is quite large, as many different inputs go through a long series of validation checks to ensure accuracy. Unfortunately, due to the time constraints I have had to remove the feature of inputting examinee and accompanist data. This was quite a difficult decision to make, but the complexity involved in this part of the process was too large for me to be able to complete it accurately. I am focussed on making the remaining inputs as robust as possible, to ensure that the system can handle a range of inputs.

The original design involved going through a series of pages to enter this data. Now, the start date will be entered by the user, and the examinee and accompanist data will be entered manually into the database before running the schedule function. As there is now only one user input (the start date), I have changed my design to promote simplicity. Clicking 'Start New Schedule' will redirect users to `schedule.php`, and this page will take the start date and validate it, with the main 'Schedule' button acting as a submit button. If validation checks are passed, the schedule will be made and then displayed at `schedule-complete.php`. This keeps the structure of the scheduling process simple, minimising the number of pages Ms Pearcey has to visit when scheduling exams. As Ms Pearcey will now manually adjust the `Examinees` and `ExamAccomps` tables, it seems sensible to give her full control over these tables. Therefore, 'Start New Schedule' will only empty `BasicInfo` and `Schedule`, not the other 2 tables.

```
<!-- enter start date -->
<div class="form-group p-3">
    <label for="basicinfo"> Enter the start date</label>
    <input type="date" class="form-control" id="basicinfo" name="basicinfo" >
</div>

<div class='fw-semibold fst-italic mt-4 p-3'>
    Please enter examinee and accompanist data directly into the database!
</div>
```

I created the interface to take this input. I also included a message to clarify how accompanist and examinee data should be entered. This message is very direct, and should clarify the situation.

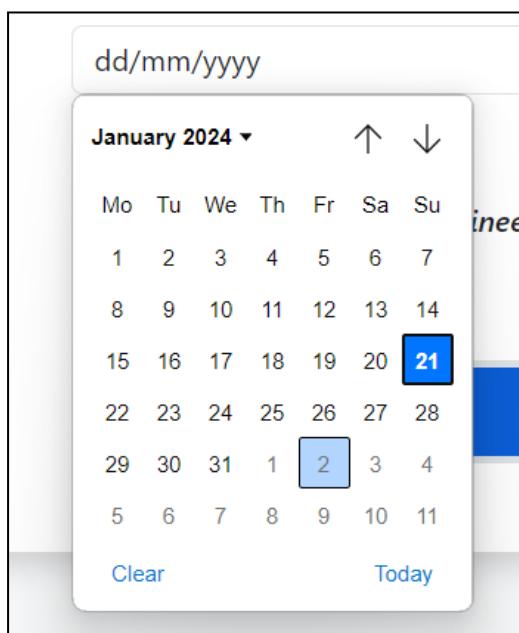
**Schedule Music Exams**

Enter the start date

📅

*Please enter examinee and accompanist data directly into the database!*

⌚ Schedule



The interface now looks like this, which I believe is quite simple and intuitive. Clicking the calendar icon brings up a calendar, allowing the user to select the required date. This is a result of the `<input type='date'>` tag. I also have refined the interface by making some slight adjustments in other areas, the most prominent being the addition of icons into the buttons in the admin dashboard.

## Schedule Music Exams

 View Existing Schedule

 Start New Schedule

**Warning:** Starting a new schedule will delete the existing one.

These icons are implemented using Bootstrap 5, and I believe that it reinforces the purpose of each button, making each one more familiar and individual. This is one of my usability features from section [2.3](#), so its implementation should make the system easier to use and more intuitive for the user. I have also displayed the start date of the exam when displaying the schedule.

```
// get start date of exam
$basicinfoSQL="SELECT StartDate FROM BasicInfo";
$basicinfoResult=$conn->query($basicinfoSQL);
$basicinfo=$basicinfoResult->fetch_assoc();
$startDate=$basicinfo['StartDate'];
// format start date appropriately
$timestamp = strtotime($startDate);
$formattedDate = date('jS M Y', $timestamp);
```

The start date is retrieved from the database. As there will only ever be one record in BasicInfo, there is no need to specify which record should be selected. The date is converted to a Unix timestamp (number of seconds since 1st Jan 1970) by strtotime(). Then, it is converted to a date in a more appropriate format that is easier to read. This date is displayed alongside the schedule, in the title.

```
// presence check
if ($startDate==""){
    $validationError="Please enter a date";
} else {
    // reject if date is in the past
    if ($startDate < $today->format('Y-m-d')){
        $validationError="The date must be in the future.";
    } else {
```

I have developed the interface to enter the date, but not yet the validation. Using the pseudocode from [2.5.4](#), I created the 2 checks shown above. This ensures that a date is

entered, and that the date is not in the past. \$today is an object of the DateTime class which stores the current date, it is formatted to be consistent with \$startDate. Helpful error messages are displayed, detailing what the issue is in each case.

```
// check if examinees is empty
$examineeSQL="SELECT * FROM examinees";
$examineeResult=$conn->query($examineeSQL);
if ($examineeResult->num_rows == 0 ){
    $validationError="There is no examinee data in the database.";
} else {
    //enter data into BasicInfo
    $enterBasicinfo="INSERT INTO basicinfo (StartDate) VALUES('$startDate')";
    $conn->query($enterBasicinfo);
    // create schedule
    [$alert,$error]=schedule($conn);
    // redirect to schedule-complete.php with messages
    $_SESSION['alert']=$alert;
    $_SESSION['error']=$error;
    header("Location:schedule-complete.php");
    exit();
}
```

I have developed one additional piece of validation: to check if the Examinees table is empty. This was not necessary before, as the CSV file of examinee information would have to be submitted. However, now that examinee information is stored directly in the database, it is possible that this table can be empty, resulting in many errors during scheduling. To improve robustness, an error is returned if the Examinees table is empty. There is no point in scheduling no examinees. However, if all examinees have no accompanist, the ExamAccomps table can be empty. Therefore, there is no validation for this table, to maximise the range of schedules that can be developed.

If all validation is passed, the entry is made into BasicInfo, and the schedule function is run. Finally, the user is taken to schedule-complete.php, and the schedule and any messages are displayed there.

### 3.11.2 Unit testing

I have integrated this module directly into the existing system, but I will just carry out testing on what has been developed in this stage. Please note that <input type='date'> will automatically prevent any invalid dates such as 50/15/2024 from being entered.

Purpose of test	Test data	Expected Result	Successful?
Testing that an empty date is rejected	NULL	“Please enter a date.”	Yes

(invalid data)			
----------------	--	--	--

## Schedule Music Exams

Please enter a date

Enter the start date

CALENDAR

*Please enter examinee and accompanist data directly into the database!*

Schedule

Purpose of test	Test data	Expected Result	Successful?
Testing that a date in the past is rejected (invalid)	01/01/2024	“The date must be in the future”	Yes

The date must be in the future.

Purpose of test	Test data	Expected Result	Successful?
Testing that the date yesterday is rejected (invalid boundary)	20/01/2024	“The date must be in the future”	Yes

The date must be in the future.

Purpose of test	Test data	Expected Result	Successful?
Testing that the date today is accepted (valid boundary)	21/01/2024	Redirected to schedule-complete.php Schedule is created and displayed	Yes

## Scheduling Results

All examinees have been scheduled!

### Exam Schedule | 21st Jan 2024

Student Name	Instrument	Grade	Accompanist Name	Start Time
Doyinsola Oliyide	Saxophone	1	Elliot Drew	09:00
Akshay Suglani	Saxophone	3	Elliot Drew	09:13
Zaid Rassam	Trumpet	1	Simon Palmer	09:26

Purpose of test	Test data	Expected Result	Successful?
Testing that the date in the future is accepted (valid)	10/02/2024	Redirected to schedule-complete.php Schedule is created and displayed	Yes

## Scheduling Results

All examinees have been scheduled!

### Exam Schedule | 10th Feb 2024

Student Name	Instrument	Grade	Accompanist Name	Start Time
Doyinsola Oliyide	Saxophone	1	Elliot Drew	09:00
Akshay Suglani	Saxophone	3	Elliot Drew	09:13
Zaid Rassam	Trumpet	1	Simon Palmer	09:26

Purpose of test	Test data	Expected Result	Successful?
Testing the validation for examinees	10/02/2024 Examinees table is empty	“There is no examinee data in the database.”	Yes

## Schedule Music Exams

There is no examinee data in the database.

Enter the start date

dd/mm/yyyy



*Please enter examinee and accompanist data directly into the database!*

⌘ Schedule

Purpose of test	Test data	Expected Result	Successful?
Testing that an exam schedule with no accompanists is produced	10/02/2024 AccomplID = null for all records in Examinees ExamAccomps is empty	Redirected to schedule-complete.php Valid schedule is created and displayed	Yes

## Exam Schedule | 26th Jan 2024

Student Name	Instrument	Grade	Accompanist Name	Start Time
Ahana Seeam	Classical Guitar	1		09:00
Aisha Nashmia	Classical Guitar	1		09:13
Aroush Haider	Classical Guitar	1		09:26
Eshal Aamir	Classical Guitar	1		09:39
Farishta Dosanjh	Classical Guitar	1		09:52
Shailey Sivathasan	Classical Guitar	1		10:05
Daniel Okpla	Classical Guitar	2		10:18
Anisha Kawle	Classical Guitar	3		10:31
<hr/>				
Sri Grandhi	Classical Guitar	3		10:59
Aakshat Kumar	Classical Guitar	4		11:12
Ananya Tare	Classical Guitar	4		11:30
Nikhil Gilliam	Classical Guitar	4		11:48
Tula Hobbs	Classical Guitar	5		12:06

This test ensures robustness in the entire system. Even with this slightly unusual input, the schedule produced is completely valid and sensible.

### 3.11.3 Review of stage 11

#### Stage 11 summary

- Due to time constraints, examinee and accompanist inputs will not be developed
- Developed the inputs for the start time with validation
- Refined the admin dashboard interface with icons
- Refined the displayed schedule with a start date

#### Testing carried out

- Used valid, invalid, and boundary data to check all possible inputs for the start time
- Included a test to ensure the system can make a schedule with no accompanists

All tests were successful in the end.

#### Success criteria met

Slight improvements to the interface have brought G1 closer to being met, by developing usability features outlined in section [2.3](#). I1 has been met in this section, as there is now an option to enter the start date of the exam when making a new schedule. There is also sufficient validation for this data.

#### Changes to the design

The main change to the design is that the feature to enter accompanist and examinee information has been removed from development. The start date has included some extra validation so that all inputs can be handled correctly.

## *Review of the project so far*

The decision to remove accompanist and examinee inputs was quite difficult, as it will force Ms Pearcey to enter this data manually into the dashboard. However, the level of complexity was simply not feasible within the available time. I chose to develop a more robust, but incomplete system over a complete, but weaker system. This means that it is still possible for this solution to be used practically without serious security issues. Furthermore, I hope to develop this feature in the near future, so developing it poorly would only result in more problems for me, and more importantly Ms Pearcey.

In addition, I have prioritised the most important parts of the system first, such as the scheduling functionality. After discussing with Ms Pearcey in section 2.5.1, we created a list of features ordered by priority. Adhering to this structured approach, I have ensured the features that are the most useful to Ms Pearcey have been developed first. This method has protected the project from potential distractions, allowing it to stay aligned with Ms Pearcey's requirements. This part of the project is quite important, but its priority falls below some other features. Therefore, the removal of this feature, while difficult, was easier than potentially making cuts to more significant modules such as the scheduling functionality.

The system is now able to take inputs to populate Students, Supervisors, and BasicInfo. Timings and Families contain static data, and Schedule is populated by the schedule function. The only 2 tables for Ms Pearcey to manually enter are Examinees and ExamAccomps.

Now that SoundBro can gather the necessary data, and produce a schedule correctly, all that is left is for the schedule to be communicated appropriately. Publishing the schedule will be the last stage of development.

My directory structure has not changed, the majority of programming in this section has been in the file schedule.php. Nevertheless, the directory structure is shown below.

```
soundbro/
|--- index.php
|--- login.php
|--- register.php
|--- register-student.php
|--- register-supervisor.php
|--- dashboard.php
|--- dashboard-admin.php
|--- schedule.php
|--- schedule-complete.php
|--- logout.php
|--- styles.css
|--- includes/
    |--- db-connect.php
    |--- navbar.php
    |--- footer.html
|--- logo/
```

```
| --- full.png  
| --- favicon3.png
```

## 3.12 Publishing the schedule

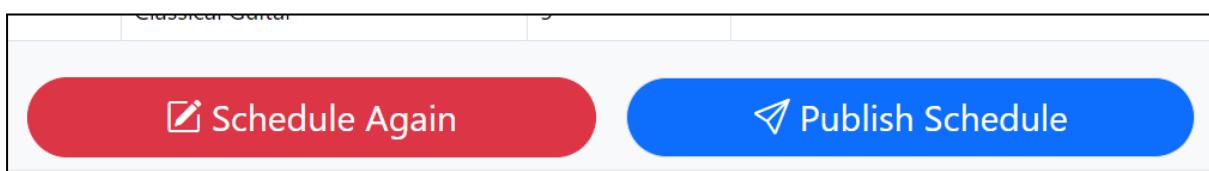
### 3.12.1 Developing

Section 2.5.5 has created several detailed algorithms for publishing the schedule, which I aim to use when developing these features. The first of these is the publish button.

#### Publish button

This button will update the value of IsPublished to 1, which is stored in BasicInfo. It will be available in ‘schedule-complete.php’. Once published, both buttons (‘Schedule Again’ and ‘Publish Schedule’) will not be available to click. A new schedule can be created in the admin dashboard only at this point. This is to simplify the page once it has been published, so that the only thing available is the schedule to view. Naturally, the publish button should disappear because something cannot be published twice.

```
<?php if ($IsPublished==0) { ?>
    <!-- schedule again -->
    <button type="submit" name="new" class="m-2 p-2 fs-4 w-25 btn btn-danger rounded-pill">
        <i class="bi bi-pencil-square"></i> Schedule Again
    </button>
    <!-- publish schedule-->
    <button type="submit" name="publish" class="fs-4 m-2 p-2 btn w-25 btn-primary border rounded-pill">
        <i class="bi bi-send"></i> Publish Schedule
    </button>
</?php } ?>
```



Using icons and colours, these buttons are emphasised and easy to press. This feature intentionally improves the usability of the system, to improve the overall user experience.

```
//publish schedule button
if(isset($_POST['publish'])){
    // set IsPublished to 1
    $sql="UPDATE basicinfo SET IsPublished=1";
    $conn->query($sql);
    //redirect to admin dashboard
    header("Location: dashboard-admin.php?published=1");
    exit();
}
```

The publish button will update the value in the database. This should cause all dashboards to display relevant data regarding their time slots (this functionality will be developed later). The user is redirected to dashboard-admin.php, along with the variable published=1. This will cause the page to display a simple success message to say that the schedule has been

published. This message should reassure Ms Pearcey by clarifying the status of the schedule.

### *displaySchedule function*

This function will be used to display the relevant data to each dashboard. Since being designed in section 2.5.5, several important events have led to some changes in how this function will be implemented.

- In stage 9, I developed an interface to display the schedule to Ms Pearcey. This was only for displaying the schedule in schedule-complete.php, and displayed the full schedule no matter what.
  - I developed this as a prototype for displaySchedule, but made some important and complex changes to the interface. For example, I split the displayed schedule up into breaks.
  - These changes have caused the program to stray quite far from the designed algorithm for displaySchedule.
- At the end of stage 10, Ms Pearcey was very pleased with this interface, saying it looked ‘clean, clear and well-presented’. She also highlighted the presentation of the timetable as ‘great’.

As Ms Pearcey is my main client, I try to cater to her needs in this project. As she was evidently very happy with the displayed schedule, I believe that there is no need to make any further changes.

This will make my solution slightly less modular, but due to the changes made in stage 9, implementing this function would be even more difficult and less sensible. This is because the function displaySchedule would have to be edited quite significantly in order to be able to display the general schedule as well as a specific schedule. To maintain simplicity, the complexities of displaying the general schedule will stay in schedule-complete.php. The function displaySchedule will only display specific schedules on the user dashboard (in dashboard.php).

Therefore, the PDF will not be made in this function, but later in this stage. This change means that I can simply display the schedule, instead of editing an array as designed in 2.5.5. It also means that ‘general’ no longer needs to be a parameter.

```
function displaySchedule($conn, $result){
```

As previously failed tests have taught me, \$conn must be passed as a parameter in order to access the database. \$result will contain a 2D associative array of the result of the query from the database. Each query will differ depending on what type of schedule is being displayed, and who it is being displayed for.

```

while ($row = $result->fetch_assoc()) {
    //get start time in correct form
    $StartTime=substr($row['StartTime'],0,5);

    // display info on table
    echo "<tr>";
    echo "<td>{$row['FirstName']} {$row['LastName']}</td>";
    $instrument=ucwords($row['Instrument']);
    echo "<td>{$instrument}</td>";
    echo "<td>{$row['Grade']}</td>";

```

The relevant fields from the result are displayed in a table format. As I have learnt from displaying the schedule in ‘schedule-complete.php’ I have formatted the start time properly to remove the number of seconds from being displayed, abstracting unnecessary data.

```

$AccompID=$row['AccompID'];
if ($AccompID){
    // if examinee has accompanist, display their name
    $accompInfoSQL="SELECT FirstName,LastName FROM Supervisors
    JOIN ExamAccomps ON Supervisors.SupervisorID = ExamAccomps.AccompID
    WHERE AccompID = '$AccompID'";
    $accompInfo=$conn->query($accompInfoSQL);
    $accompRow=$accompInfo->fetch_assoc();
    echo "<td>{$accompRow['FirstName']} {$accompRow['LastName']}</td>";
} else {
    // empty cell to maintain format
    echo "<td> </td>";
}
//display start time
echo "<td><strong>{$StartTime}</strong></td>";
echo "</tr>";

```

Just as designed, my function will check if AccompID is null or not. If it is not null, the accompanist’s name is retrieved and displayed. If so, then an empty cell is displayed to maintain the regular format of the table. The start time is also displayed in bold. The table will be started and ended outside of this function, as the table may also display certain messages, such as ‘You are not scheduled to play in the upcoming exams.’ This function will just display the important information of the schedule, it will only be called if the program is sure that the user is involved in the schedule.

### *Displaying relevant information on the dashboard*

This section works in conjunction with the displaySchedule function. The designed pseudocode will help me to achieve success criteria L7-10. I realised one thing while developing this function, that when teachers view the schedule, they will not view their own name. Therefore, I will edit the table so that the teacher name is displayed, but **only** for that

table. This means that several FirstNames and LastNames will be returned, so they need to be distinguished in the SQL query, by returning student\_fname and teacher\_fname, and the same for last names. This prevents confusion between the names. Also, displaySchedule needs to be edited to allow the teacher name to be displayed.

```
if ($teacher){  
| echo "<td><strong>{$row['teacher_fname']} {$row['teacher_lname']}</strong></td>";  
}|
```

\$teacher is another parameter to allow the function to distinguish whether or not to display the teacher's name. It is displayed in bold for emphasis.

I retrieved the value of IsPublished and the start date from BasicInfo at the start. The pseudocode specifies that the first if statement should check if the schedule is published or not. However, I think it would be more appropriate to check the user type first, as it simplifies how messages are displayed, such as the results\_message (see below).

```
if ($_SESSION['user_type']=='student'){  
| $results_message="If you have recently taken an exam, results will be displayed on the  
| music notice board, or you will be contacted by your music teacher.";  
| //is schedule published yet  
| if ($IsPublished==1){ ?>  
| | <!-- display exam schedule -->
```

After checking if the user is a student, the value of IsPublished is then checked. If it has been published, a table is started using the `<table>` tag, in the same style as `schedule-complete.php`. The main body of the table is shown below.

```
<tbody>  
| <?php  
| // get info from DB to create correct schedule format  
| $examineeSQL="SELECT students.FirstName AS student_fname,  
| students.LastName AS student_lname,Instrument,Grade,StartTime,examinees.AccompID  
| FROM schedule  
| JOIN examinees ON schedule.ExamineeID=examinees.ExamineeID  
| JOIN students ON students.StudentID=examinees.ExamineeID  
| WHERE examinees.ExamineeID='$ID';"  
| $examineeResult = $conn->query($examineeSQL);  
  
| if ($examineeResult and $examineeResult->num_rows > 0) {  
| | //display schedule  
| | displaySchedule($conn,$examineeResult,false);  
| } else {  
| | echo "<tr><td colspan='5'>  
| | You aren't scheduled to play in the upcoming exams.<br>$results_message</td></tr>";  
| }  
| ?>  
</tbody>
```

This SQL query was quite difficult to create, so I tested it and found the correct query by executing the query in phpMyAdmin. I did the same for all SQL queries in this section. The \$ID is the same as the session variable ‘user\_id’. As long as the result returns at least 1 row, displaySchedule is called to display the schedule. If the current user is not going to play in the upcoming exams, the query will return an empty result, so the number of rows will be zero. In this case, the body of the table will display a message to clarify the situation, as well as a message about how results will be communicated. If, for any reason, the query does not execute correctly, the \$examineeResult will be false. In this case, it will display the same message. This additional validation prevents large error messages from being displayed.

The table is then ended using HTML.

```
} else {  
    // not published yet  
    $message.= "Nothing has been scheduled yet. Check again later.<br>$results_message";  
}
```

If IsPublished is 0, then the appropriate message is displayed on the student’s dashboard. If the user\_type is not a student, it must be a supervisor or admin. Just as the pseudocode has done, there is no need to check the user type again (as both supervisor and admin will view the same style of timetable in their dashboard), we can simply use ‘else’.

First, the teacher’s timetable is made. If IsPublished is 0, the same message as above is displayed, just without the results message. If IsPublished is 1, the program tries to display a teacher schedule and accompanist schedule. These are 2 separate tables to indicate the 2 separate roles a supervisor can have.

```

<tbody>
    <?php
        // get info from DB to create correct schedule format
        $teacherSQL="SELECT students.FirstName AS student_fname,students.LastName
        AS student_lname,Instrument,Grade,StartTime,examinees.AccompID,ScheduleID,
        supervisors.FirstName AS teacher_fname,supervisors.LastName AS teacher_lname
        FROM schedule
        JOIN examinees ON schedule.ExamineeID=examinees.ExamineeID
        JOIN supervisors ON supervisors.SupervisorID=examinees.SupervisorID
        JOIN students ON students.StudentID=examinees.ExamineeID
        WHERE supervisors.SupervisorID='$ID'
        ORDER BY StartTime ASC";
        $teacherResult = $conn->query($teacherSQL);
        // if result is not empty
        if ($teacherResult and $teacherResult->num_rows > 0) {
            // display schedule
            displaySchedule($conn,$teacherResult,true);
        } else {
            echo "<tr><td colspan='6'>
                You aren't teaching any upcoming examinees.</td></tr>";
        }
    ?>
</tbody>

```

This is the body of the table for the teacher schedule. A complex SQL query is made to retrieve all scheduling data. I found that using my entity relationship diagram ([2.4.1](#)) was a big help in constructing these queries, throughout the entire development process. One intricate detail in this query is that ‘Examinees’ is joined directly to ‘Supervisors’ using the common SupervisorID, so that all teachers can be accessed in the database. Again, if the result is empty, an appropriate message is displayed. If not, displaySchedule is called, and this time \$teacher is true. This will cause the table to display the teacher’s name as well.

```

<tbody>
    <?php
        // get info from DB to create correct schedule format
        $accompSQL="SELECT students.FirstName AS student_fname,students.LastName
        AS student_lname,Instrument,Grade,StartTime,examinees.AccompID,
        examinees.ExamineeID,ScheduleID
        FROM schedule
        JOIN examinees ON schedule.ExamineeID=examinees.ExamineeID
        JOIN examaccompson ON examinees.AccompID=examaccompson.AccompID
        JOIN supervisors ON examaccompson.AccompID=supervisors.SupervisorID
        JOIN students ON students.StudentID=examinees.ExamineeID
        WHERE supervisors.SupervisorID='\$ID'
        ORDER BY StartTime ASC";
        $accompResult = $conn->query($accompSQL);
        // if result is not empty
        if ($accompResult and $accompResult->num_rows > 0) {
            // display schedule
            displaySchedule($conn,$accompResult,false);
        } else {
            echo "<tr><td colspan='5'>
            You aren't accompanying any upcoming exams.</td></tr>";
        }
    ?>
</tbody>

```

The next table to be displayed is the accompanist schedule. This will only be displayed if the schedule has been published, like all other tables. The important difference here is that the Supervisors table is connected to the Examinees table indirectly, it is connected **through ExamAccomps**. This difference means that the only supervisors to return a result will be those who exist in all 3 tables: ExamAccomps, Supervisors, and Examinees. These are the accompanists, not the teachers or anyone else, so the query should correctly display schedule information for just the accompanists. The function is then called, as long as the result is not empty. This time, \$teacher is false.

Finally, the relevant messages that were previously stored in \$message are displayed, along with a message to contact Ms Pearcey if there are any concerns. I have displayed a horizontal line using `<hr>` to break up the interface and emphasise the final message. I have also emphasised this message by using a larger font size. I want this message to stand out so users feel comfortable and encouraged to talk to Ms Pearcey if they have any issues. My examinee survey during analysis section [1.3.2](#) found that some students did not speak to Ms Pearcey even though they had concerns, which may have led to poor exam performance. This message is emphasised to prevent this from happening again.

```

<p class="text-center">
<?php echo $message."<br>";
if ($message != ""){ ?>
|   <hr>
<?php } ?>
|   <p class="fs-5 text-center">
|       Email Ms. Pearcey at <strong>admin@email.com</strong> if you have any concerns.
|   </p>
</p>

```

This should now be able to correctly display the relevant information about the schedule for all users at any time.

### *Making the PDF*

I attempted to use DOMPDF to create a PDF of the final schedule. However, I ran into lots of errors when trying to use this library. Unfortunately, the schedule started to be displayed on the second page, and stopped abruptly. It seemed that it was not compatible with the styling used, so I removed some styling and tried again. This had no impact. I researched online, read through the documentation and tried editing different aspects of the program. Unfortunately, none of these techniques worked and, due to the time constraints, I have had to remove this feature.

I could have tried using another library, such as TCPDF. However, the problem was becoming quite complex and, given the time available, there will be no functionality to download a PDF of this schedule.

Exam Schedule				
Student Name	Instrument	Grade	Accompanist Name	Start Time
Doyinsola Oliyide	Saxophone	1	Elliot Drew	09:00
Akshay Suglani	Saxophone	3	Elliot Drew	09:13
Zaid Rassam	Trumpet	1	Simon Palmer	09:26
Jiayi (Eva) Chen	Clarinet	3	John Meadows	09:39
Henna Naveed	Clarinet	5	John Meadows	09:52

An example of one of the failed PDFs is shown above. If I had more time, this feature would definitely be attempted again, trying other libraries using more specialised debugging techniques.

### 3.12.2 Unit testing

Purpose of test	Test data	Expected Result	Successful?
Publish button should update the BasicInfo table	Schedule is made, then 'publish schedule' is clicked	Redirected to dashboard-admin.php "The schedule has been published" is displayed IsPublished field in BasicInfo is set to 1	Yes

StartDate	Timestamp	IsPublished
2024-01-27	2024-01-21 18:25:44	1

Purpose of test	Test data	Expected Result	Successful?
Students view relevant information of their schedule	Student (zaid.rassam@example.com) logs in after schedule is published  This student has been scheduled	Single time slot displayed for Zaid Rassam  Message to email Ms Pearcey	Yes

## Dashboard

Welcome, Zaid!

### Exam Schedule | 1st Feb 2024

Student Name	Instrument	Grade	Accompanist Name	Start Time
Zaid Rassam	Trumpet	1	Simon Palmer	09:26

Email Ms. Pearcey at [admin@email.com](mailto:admin@email.com) if you have any concerns.

Purpose of test	Test data	Expected Result	Successful?
Acccompanists view relevant information of their schedule	<p>Supervisor (<a href="mailto:simon.palmer@example.com">simon.palmer@example.com</a>) logs in after schedule is published</p> <p>He has been scheduled as an accompanist only</p>	<p>Teacher table: 'You aren't teaching any upcoming examinees'</p> <p>Accompanist table: all of Simon Palmer's exams displayed</p> <p>Message to email Ms Pearcey</p>	Yes

## Dashboard

Welcome, Simon!

### Exam Schedule | 1st Feb 2024 | Teacher Info

Student Name	Instrument	Grade	Teacher Name	Accompanist Name	Start Time
You aren't teaching any upcoming examinees.					

### Exam Schedule | 1st Feb 2024 | Accompanist Info

Student Name	Instrument	Grade	Accompanist Name	Start Time
Zaid Rassam	Trumpet	1	Simon Palmer	09:26
Leya Rahman	Trumpet	5	Simon Palmer	10:10
Xander Davis	Trumpet	6	Simon Palmer	10:28
Lily-Marie Le Blanc	Viola	3	Simon Palmer	11:06
Milaura Fernando	Violin	1	Simon Palmer	11:19
Maryam Rahman	Violin	5	Simon Palmer	11:32
Aliza Rayhan	Flute	6	Simon Palmer	11:50
Lauren Pumphrey	Flute	6	Simon Palmer	12:13

Email Ms. Pearcey at [admin@email.com](mailto:admin@email.com) if you have any concerns.

Purpose of test	Test data	Expected Result	Successful?
Teachers view relevant information of their schedule	<p>Supervisor (john.phillips@example.com) logs in after schedule is published</p> <p>He has been scheduled as a teacher only</p>	<p>Teacher table: All of John Phillips' students are displayed. John Phillips is displayed in an additional 'Teacher Name' column</p> <p>Accompanist table: "You aren't accompanying any upcoming exams"</p> <p>Message to email Ms Pearcey</p>	Yes

## Dashboard

Welcome, John!

### Exam Schedule | 1st Feb 2024 | Teacher Info

Student Name	Instrument	Grade	Teacher Name	Accompanist Name	Start Time
Ahana Seeam	Classical Guitar	1	John Phillips		13:36
Aisha Nashmia	Classical Guitar	1	John Phillips		13:49
Aroush Haider	Classical Guitar	1	John Phillips		14:02
Eshal Aamir	Classical Guitar	1	John Phillips		14:15
Farishta Dosanjh	Classical Guitar	1	John Phillips		14:28
Shailey Sivathasan	Classical Guitar	1	John Phillips		14:41
Anisha Kawle	Classical Guitar	3	John Phillips		15:07
Ananya Tare	Classical Guitar	4	John Phillips		16:06
Tula Hobbs	Classical Guitar	5	John Phillips		16:42

### Exam Schedule | 1st Feb 2024 | Accompanist Info

Student Name	Instrument	Grade	Accompanist Name	Start Time
You aren't accompanying any upcoming exams.				

Email Ms. Pearcey at [admin@email.com](mailto:admin@email.com) if you have any concerns.

Purpose of test	Test data	Expected Result	Successful?
Supervisors view relevant information of their schedule	Supervisor (john.meadows@example.com) logs in after schedule is published	Teacher table: All of John Phillips' students are displayed	Yes

People who both teach and accompany exams	published He has been scheduled as an accompanist and teacher	Accompanist table: All of John Meadows' exams are displayed  Message to email Ms Pearcey	
---	--	--	--

**Dashboard**

Welcome, John!

**Exam Schedule | 1st Feb 2024 | Teacher Info**

Student Name	Instrument	Grade	Teacher Name	Accompanist Name	Start Time
Jiayi (Eva) Chen	Clarinet	3	John Meadows	John Meadows	09:39
Henna Naveed	Clarinet	5	John Meadows	John Meadows	09:52

**Exam Schedule | 1st Feb 2024 | Accompanist Info**

Student Name	Instrument	Grade	Accompanist Name	Start Time
Jiayi (Eva) Chen	Clarinet	3	John Meadows	09:39
Henna Naveed	Clarinet	5	John Meadows	09:52

Email Ms. Pearcey at [admin@email.com](mailto:admin@email.com) if you have any concerns.

The same exams are repeated, but I believe that this clarifies the 2 different roles Mr Meadows has in these exams. Combining this into one table may lead to confusion as to whether or not he is an accompanist for these exams.

Purpose of test	Test data	Expected Result	Successful?
Supervisor view if they are not involved at all in the upcoming schedule	Supervisor (alex.tune@example.com) logs in after schedule is published  She is not a teacher or accompanist for any examinees in the upcoming schedule.	Teacher table: 'You aren't teaching any upcoming examinees'  Accompanist table: "You aren't accompanying any upcoming exams"  Message to email Ms Pearcey	Yes

### Exam Schedule | 27th Jan 2024 | Teacher Info

Student Name	Instrument	Grade	Teacher Name	Accompanist Name	Start Time
You aren't teaching any upcoming examinees.					

### Exam Schedule | 27th Jan 2024 | Accompanist Info

Student Name	Instrument	Grade	Accompanist Name	Start Time
You aren't accompanying any upcoming exams.				

Email Ms. Pearcey at [admin@email.com](mailto:admin@email.com) if you have any concerns.

Purpose of test	Test data	Expected Result	Successful?
Student view if they are not involved at all in the upcoming schedule	<p>Student (eddie.greenwood@example.com) logs in after schedule is published</p> <p>He is not going to play in the upcoming exam</p>	<p>You aren't scheduled to play in the upcoming exams.</p> <p>Results message</p> <p>Message to email Ms Pearcey</p>	Yes

### Exam Schedule | 27th Jan 2024

Student Name	Instrument	Grade	Accompanist Name	Start Time
You aren't scheduled to play in the upcoming exams.				

If you have recently taken an exam, results will be displayed on the music notice board, or you will be contacted by your music teacher.

Email Ms. Pearcey at [admin@email.com](mailto:admin@email.com) if you have any concerns.

Purpose of test	Test data	Expected Result	Successful?
Ms Pearcey's (admin) view of dashboard after publishing	<p>Ms Pearcey (admin@email.com) logs in after publishing the exam</p> <p>She is not going to teach or accompany any examinees</p>	<p>Link to go to the admin dashboard</p> <p>Teacher table: 'You aren't teaching any upcoming examinees'</p> <p>Accompanist table: "You aren't accompanying any upcoming exams"</p> <p>Message to email Ms Pearcey</p>	Yes

# Dashboard

Welcome, Lorne!

Welcome! If you are here for admin purposes, click [here](#).  
If not, stay on this page to see music teacher / accompanist info.

## Exam Schedule | 27th Jan 2024 | Teacher Info

Student Name	Instrument	Grade	Teacher Name	Accompanist Name	Start Time
You aren't teaching any upcoming examinees.					

## Exam Schedule | 27th Jan 2024 | Accompanist Info

Student Name	Instrument	Grade	Accompanist Name	Start Time
You aren't accompanying any upcoming exams.				

Email Ms. Pearcey at [admin@email.com](mailto:admin@email.com) if you have any concerns.

Purpose of test	Test data	Expected Result	Successful?
Student view when BasicInfo is empty	Student (zaid.rassam@example.com) logs in after 'Start New Schedule' is clicked.	"Nothing has been scheduled yet. Check again later."  Results message.  Message to email Ms Pearcey	No

**( ! )** Warning: Trying to access array offset on value of type null in C:\wamp64\www\soundbro\dashboard.php on line 96

Call Stack				
#	Time	Memory	Function	Location
1	0.0001		362744 {main}()	...\dashboard.php:0

**( ! )** Warning: Trying to access array offset on value of type null in C:\wamp64\www\soundbro\dashboard.php on line 102

Call Stack				
#	Time	Memory	Function	Location
1	0.0001		362744 {main}()	...\dashboard.php:0

of type null in C:\wamp64\www\soundbro\dashboard.php on line 102

Location	
)	...\dashboard.php:0

Nothing has been scheduled yet. Check again later.

If you have recently taken an exam, results will be displayed on the music notice board, or you will be contacted by your music teacher.

Email Ms. Pearcey at [admin@email.com](mailto:admin@email.com) if you have any concerns.

The correct outputs have been displayed, but an unexpected warning has been returned as well. It points out lines 96 and 102. These 2 lines retrieve the fields StartDate and IsPublished. The table BasicInfo is empty, so there is nothing to be retrieved and therefore the warning is shown. This has a simple solution: only retrieve the values if the result has more than one row. The logic of displaying messages must be updated as well to reflect this change.

```
// get basic info + start date
$publishSQL="SELECT IsPublished,StartDate FROM BasicInfo";
$publishResult=$conn->query($publishSQL);
if ($publishResult and $publishResult->num_rows > 0){
    $publishInfo=$publishResult->fetch_assoc();
    $IsPublished=$publishInfo['IsPublished'];
    $startDate=$publishInfo['StartDate'];
    $timestamp = strtotime($startDate);
    $formattedDate = date('jS M Y', $timestamp);
} else {
    // schedule not created yet, so set IsPublished to 0
    $IsPublished = 0;
}
```

Previously, all data was retrieved without any if statements. Now, data is only retrieved if the result is not empty. This should prevent the warning. If the result is empty, IsPublished is set to 0. Without this declaration, the variable would not be set, and an error would be returned when checking if IsPublished == 1. Making it 0 will ensure that the correct messages are

displayed. It also makes more logical sense, if a schedule has not been made, it has not been published, so IsPublished = 0. I ran this test again to check if the error has been fixed.

Purpose of test	Test data	Expected Result	Successful?
Student view when BasicInfo is empty	Student (zaid.rassam@example.com) logs in after 'Start New Schedule' is clicked.	"Nothing has been scheduled yet. Check again later."  Results message.  Message to email Ms Pearcey	Yes

# Dashboard

Welcome, Zaid!

---

Nothing has been scheduled yet. Check again later.  
If you have recently taken an exam, results will be displayed on the music notice board, or you will be contacted by your music teacher.

---

Email Ms. Pearcey at **admin@email.com** if you have any concerns.

Purpose of test	Test data	Expected Result	Successful?
Supervisor view when BasicInfo is empty	Supervisor (simon.palmer@example.com) logs in after 'Start New Schedule' is clicked.	"Nothing has been scheduled yet. Check again later."  Message to email Ms Pearcey	Yes

# Dashboard

Welcome, Simon!

---

Nothing has been scheduled yet. Check again later.

---

Email Ms. Pearcey at **admin@email.com** if you have any concerns.

Purpose of test	Test data	Expected Result	Successful?
Student view when schedule is created, but not published	Student (zaid.rassam@example.com) logs in after schedule is made, but before schedule is published	<p>“Nothing has been scheduled yet. Check again later.”</p> <p>Results message.</p> <p>Message to email Ms Pearcey</p>	Yes

Nothing has been scheduled yet. Check again later.  
 If you have recently taken an exam, results will be displayed on the music notice board, or you will be contacted by your music teacher.

Email Ms. Pearcey at **admin@email.com** if you have any concerns.

Purpose of test	Test data	Expected Result	Successful?
Supervisor view when schedule is created, but not published	Supervisor (simon.palmer@example.com) logs in after schedule is made, but before schedule is published	<p>“Nothing has been scheduled yet. Check again later.”</p> <p>Message to email Ms Pearcey</p>	Yes

Nothing has been scheduled yet. Check again later.

Email Ms. Pearcey at **admin@email.com** if you have any concerns.

### 3.12.3 Review of stage 12

#### Stage 12 summary

- Created the ‘Publish Schedule’ button
- Created the ‘displaySchedule’ function
- Called this function to display specific and relevant information to each user
- The PDF functionality has been removed

#### Testing carried out

- Tested the functionality of the “Publish Schedule” button
- Tested what students and supervisors see on their dashboard when:
  - A new schedule is started (ie. the old schedule has just been deleted)
  - A schedule is created but not published
  - A schedule is published, and they are in the schedule

- A schedule is published, and they are not in the schedule
- Tested what Ms Pearcey sees on her dashboard

All tests were successful in the end.

### *Success criteria met*

Now that the development is complete, I believe that G1 has been met, as the entire interface is consistently simple and easy to navigate. Post-development testing will check this.

This section has met success criteria L7-10, as all users are now able to log on and view their time slot(s), or any important messages. They can also see a message to email Ms Pearcey. All of these helpful messages have been carefully designed to prevent confusion, reassure users, and encourage communication. All the time slots displayed are completely accurate, they are relevant and specific to the user, and they do not contain any more or any less data than needed.

### *Changes to the design*

The major change to the design is that the PDF feature has been removed. This corresponds to 3.3 in the structure diagram. I would have liked to develop this successfully, but had to remove this feature due to the time constraints compared to the surprising complexity of the feature.

Additionally, I have included some more usability features to make the system more intuitive. After publishing the schedule, there is a success message to say that the schedule has been published. Also, the table for teacher information has included an extra column, the teacher's name. I think that this clarifies what each person's role is in the exam to prevent misunderstanding or miscommunication. Messages like these are simple but thoughtful, and constantly allow the user to feel comfortable with the system, as designed in section [2.3](#).

After Ms Pearcey praised the interface for the schedule ([3.10.2](#)), I decided not to change it any more. In fact, I decided to replicate this style when displaying the table on the dashboard for every user. This led to changes in the displaySchedule function compared to its designed pseudocode, but its main functionality is still the same as originally designed. This decision gives me confidence that the interface is clean and clear throughout the system.

### *Review of the project so far*

The development stage is now complete. It has a clear and simple interface, which is integrated with a fully functional login system and scheduling feature. It takes the input of a StartDate, and displays the produced schedule intuitively and effectively. The schedule can be published so that all users can see exactly when their exam is. Throughout the system, helpful messages are displayed on the screen to direct the user in the right direction, preventing them from feeling overwhelmed. All pages link together intuitively, making it easy to navigate to the right page, and logins provide added security to ensure that only the admin can schedule exams.

Although some features have not been developed, I am quite pleased with the current software developed. The most important part of the software, the scheduling functionality, seems to be working very well. Unit and system testing have shown that the system is robust, usable and functional. I am now ready to present this product to my stakeholders for their final feedback, which will help me to complete the post-development testing effectively.

The final directory structure is shown below. Stage 12 has added the publish button to schedule-complete.php, and made significant advancements to dashboard.php by displaying specific data for each user.

```
soundbro/
|--- index.php
|--- login.php
|--- register.php
|--- register-student.php
|--- register-supervisor.php
|--- dashboard.php
|--- dashboard-admin.php
|--- schedule.php
|--- schedule-complete.php
|--- logout.php
|--- styles.css
|--- includes/
    |--- db-connect.php
    |--- navbar.php
    |--- footer.html
|--- logo/
    |--- full.png
    |--- favicon3.png
```

# 4 Post Development Testing

## 4.1 Interview with Ms Pearcey

In order to properly test and evaluate my solution, I had a 45 minute interview with Ms Pearcey, in which I showed her the finished software and had a discussion about each aspect of the system. Ms Pearcey is my client, and throughout this project I have aimed to develop a solution that reduces her workload, and caters to her requirements. Therefore, getting her feedback is a natural part of this process, and will help me to understand which parts of the solution are good, and which parts need improvement.

As planned in section [2.7](#), I have saved 2 sets of real exam data for these tests. These are the Summer 2023 and Spring 2023 exam cycles. To save Ms Pearcey's time, I created new databases called db\_summer\_23, db\_spring\_23, and db\_winter\_23 for each of these exam cycles. The database of Winter 2023 was already populated correctly from the system test that used this data in stage 10 ([3.10.2](#)). By taking the spreadsheet of each cycle's data, I populated Students, Supervisors, Examinees, and ExamAccomps. Families and Timings contain static data, so did not change at all. This allowed Ms Pearcey to go through the entire process of scheduling exams, from login to publishing, without having to manually enter the data in the database. I informed her of this process during our discussion.

I showed Ms Pearcey the produced schedules for Spring 2023 and Summer 2023 in this interview, and she provided feedback. I also showed her the Winter 2023 schedule to get some more detailed feedback. By showing all 3 exam cycles, my sample of real data was as representative as possible. The variations within each set of data led to different schedules being produced, and receiving feedback for all gives me the best chance of assessing how this software would work in real life.

The full interview transcript can be found in the [Appendix](#), but a summary is provided below. During this interview, I showed Ms Pearcey the system on my laptop, so the transcript may be unclear at times without the context of what I was showing her. I will reference this interview throughout my testing and evaluation sections.

### *Spring 2023 schedule*

The produced schedule is shown below. This should put Ms Pearcey's comments into better context.

## Exam Schedule | 1st Mar 2024

Student Name	Instrument	Grade	Accompanist Name	Start Time
Sam Nouhov	Jazz Saxophone	6	Elliot Drew	<b>09:00</b>
Daniel Okpla	Classical Guitar	1		<b>09:23</b>
Olivia Yang	Classical Guitar	2		<b>09:36</b>
Reena Koiri	Classical Guitar	2		<b>09:49</b>
Maryam Mohamed	Clarinet	5	John Meadows	<b>10:02</b>
Ella Goldberg	Clarinet	6	John Meadows	<b>10:20</b>
Charlotte Wood	Clarinet	7	John Meadows	<b>10:58</b>
Matteo Gianni	Baritone	6	Simon Palmer	<b>11:21</b>
Clover Webster	Cornet	5	Simon Palmer	<b>11:44</b>
Edward Cheung	French Horn	2	Simon Palmer	<b>12:02</b>
Basheir Said	Trumpet	1	Simon Palmer	<b>12:15</b>
Lemuel Adjei	Trumpet	3	Simon Palmer	<b>13:28</b>
Kirsty Jenkins	Clarinet	7	John Meadows	<b>13:41</b>
Benjamin Law	Trumpet	6	Simon Palmer	<b>14:04</b>
Shreya Jondhale	Violin	5	Simon Palmer	<b>14:27</b>
William Lin	Violin	5	Simon Palmer	<b>14:45</b>
Anupama Harish	Violin	6	Simon Palmer	<b>15:18</b>
Anushtup Chatterjee	Violin	6	Simon Palmer	<b>15:41</b>
Ananya Tare	Classical Guitar	3		<b>16:04</b>
Alicia Dieu	Classical Guitar	3		<b>16:17</b>
Levi Dandy	Classical Guitar	3		<b>16:30</b>
Sana Khan	Classical Guitar	3		<b>16:43</b>

Figure 4.1a: Schedule produced using Spring 2023 data

Ms Pearcey said that the schedule produced was very good, and when I explained how accompanists are sorted by free time, she said ‘I love that idea’. Due to a gap in availability, the software has scheduled 3 unaccompanied guitar exams in the morning. Ms Pearcey was especially happy to see this, as she suggested this functionality at the start of this project. During our stakeholder interview, she mentioned this technique as something that helps her to schedule exams efficiently. Therefore, seeing it implemented on real data is a great sign.

She singled out the 13:41 exam for Kirsty Jenkins as the only issue in the schedule. This exam would be “the only thing that would be problematic in real life”, because the accompanist, John Meadows, would prefer to have all of his exams together. It would be better if the exam started at 11:21.

### *Summer 2023 schedule*

I then showed her the schedule that was produced for the Summer 2023 exam cycle.

Exam Schedule   1st Jul 2024				
Student Name	Instrument	Grade	Accompanist Name	Start Time
Marriam Javed	Trombone	1	Simon Palmer	<b>09:00</b>
Deeksha Sharma	Trombone	4	Simon Palmer	<b>09:13</b>
Aman Koiri	Clarinet	5	John Meadows	<b>09:31</b>
Clara Hilton-Widdows	Clarinet	5	John Meadows	<b>09:49</b>
Rinsola Alatise	Clarinet	5	John Meadows	<b>10:07</b>
Samia Islam	Clarinet	5	John Meadows	<b>10:25</b>
<hr/>				
Inaaya Kassim	Violin	1	Simon Palmer	<b>10:58</b>
Birle Tenekeci	Violin	2	Simon Palmer	<b>11:11</b>
Louisa Dilling	Violin	6	Simon Palmer	<b>11:24</b>
Tamsin Howard	Violin	7	Simon Palmer	<b>11:47</b>
Diya Sharath	Classical Guitar	1		<b>12:10</b>
Lucas Evans	Classical Guitar	1		<b>12:23</b>
<hr/>				
Catherine Mae Villabroza	Jazz Saxophone	7	Elliot Drew	<b>13:36</b>

Ryheem Miah	Saxophone	1	Elliot Drew	<b>13:59</b>
Mikayla Rodney	Saxophone	3	Elliot Drew	<b>14:12</b>
Mithun Kesavan	Classical Guitar	1		<b>14:25</b>
Sara Sadiq	Classical Guitar	1		<b>14:38</b>
Zubair Ahmed	Classical Guitar	1		<b>14:51</b>
Zaynah Jamal	Classical Guitar	2		<b>15:04</b>
Aakshat Kumar	Classical Guitar	3		<b>15:32</b>
Chardé Levermore	Classical Guitar	3		<b>15:45</b>
Sana Khan	Classical Guitar	4		<b>15:58</b>
Anesha Mitra	Classical Guitar	5		<b>16:16</b>
Elizabeth Shpectorov	Classical Guitar	7		<b>16:34</b>

Figure 4.1b: Schedule produced using Summer 2023 data

Ms Pearcey was even happier with this schedule, saying that it is ‘ideal’. She highlighted the effectiveness of how accompanist grouping was balanced with instrument grouping, saying that she finds this especially difficult but my system has handled it well. When I asked “Is this how you would schedule exams”, she mentioned that the only possible improvement would be to change the first 2 exams to unaccompanied guitars, which would move all of Mr Palmer’s exams into one block to save time.

### *Winter 2023 schedule*

Here is the schedule produced using the data for the Winter 2023 exam cycle.

## Exam Schedule | 24th Jan 2024

Student Name	Instrument	Grade	Accompanist Name	Start Time
Doyinsola Oliyide	Saxophone	1	Elliot Drew	<b>09:00</b>
Akshay Suglani	Saxophone	3	Elliot Drew	<b>09:13</b>
Zaid Rassam	Trumpet	1	Simon Palmer	<b>09:26</b>
Jiayi (Eva) Chen	Clarinet	3	John Meadows	<b>09:39</b>
Henna Naveed	Clarinet	5	John Meadows	<b>09:52</b>
Leya Rahman	Trumpet	5	Simon Palmer	<b>10:10</b>
Xander Davis	Trumpet	6	Simon Palmer	<b>10:28</b>
<hr/>				
Lily-Marie Le Blanc	Viola	3	Simon Palmer	<b>11:06</b>
Milaura Fernando	Violin	1	Simon Palmer	<b>11:19</b>
Maryam Rahman	Violin	5	Simon Palmer	<b>11:32</b>
Aliza Rayhan	Flute	6	Simon Palmer	<b>11:50</b>
Lauren Pumphrey	Flute	6	Simon Palmer	<b>12:13</b>
<hr/>				
Ahana Seeam	Classical Guitar	1		<b>13:36</b>
<hr/>				
Aisha Nashmia	Classical Guitar	1		<b>13:49</b>
Aroush Haider	Classical Guitar	1		<b>14:02</b>
Eshal Aamir	Classical Guitar	1		<b>14:15</b>
Farishta Dosanjh	Classical Guitar	1		<b>14:28</b>
Shailey Sivathasan	Classical Guitar	1		<b>14:41</b>
Daniel Okpla	Classical Guitar	2		<b>14:54</b>
Anisha Kawle	Classical Guitar	3		<b>15:07</b>
<hr/>				
Sri Grandhi	Classical Guitar	3		<b>15:35</b>
Aakshat Kumar	Classical Guitar	4		<b>15:48</b>
Ananya Tare	Classical Guitar	4		<b>16:06</b>
Nikhil Gilliam	Classical Guitar	4		<b>16:24</b>
Tula Hobbs	Classical Guitar	5		<b>16:42</b>

Figure 4.1c: Schedule produced using Winter 2023 data

Ms Pearcey said that this was even better, as there was an even shorter gap between Mr Palmer's exams. The schedule looked good in all respects. She said that she would have scheduled a grade 1 guitar instead of Zaid Rassam at 09:26, so that Mr Palmer would have all exams together. But she then realised that this would cause his exams to spill over past lunch, which would be worse than the current schedule. Therefore, she concluded that the produced schedule is completely fine with her, she would not want to make any changes at all.

Ms Pearcey also shared the schedules she made for each exam cycle. She stressed that the schedule doesn't have to be exactly the same as hers, as there are many different possible schedules that could be just as good. Therefore, I only asked her to compare the Winter 2023 schedule to hers. I did not ask her to do this for the other schedules. The schedule she produced is shown below.

## Wednesday 22<sup>nd</sup> November 2023

Forename	Surname	Instrument	Grade	Time
Doyinsola	Oliyide	Saxophone	1	9:00
Akshay	Suglani	Saxophone	3	9:13
Aliza	Rayhan	Flute	6	9:26
Lauren	Pumphrey	Flute	6	9:49
Lily-Marie	Le Blanc	Viola	3	10:12
Milaura	Fernando	Violin	1	10:40
Maryam	Rahman	Violin	5	10:53
Zaid	Rassam	Trumpet	1	11:11
Leya	Rahman	Trumpet	5	11:24
Xander	Davis	Trumpet	6	11:42
Jiayi (Eva)	Chen	Clarinet	3	12:04
Henna	Naveed	Clarinet	5	12:18
Ananya	Tare	Classical Guitar	4	13:36
Aisha	Nashmia	Classical Guitar	1	13:54
Farishta	Dosanjh	Classical Guitar	1	14:07
Eshal	Aamir	Classical Guitar	1	14:20
Shailey	Sivathasan	Classical Guitar	1	14:33
Ahana Aditi	Seeam	Classical Guitar	1	14:46
Aroush	Haider	Classical Guitar	1	14:59
Hidayah	Saqib	Classical Guitar	2	15:27
Anisha	Kawle	Classical Guitar	3	15:40
Tula	Hobbs	Classical Guitar	5	15:53
Nikhil	Gilliam	Classical Guitar	4	16:11
Daniel	Okpla	Classical Guitar	2	16:29
Sri	Grandhi	Classical Guitar	3	16:42
Aakshat	Kumar	Classical Guitar	4	16:55

Figure 4.1d: Ms Pearcey's schedule for Winter 2023

**Please note: The data Ms Pearcey provided did not include the student Hidayah Saqib.** All other data is the same. Therefore, the schedule here is slightly longer than mine. I showed both schedules side by side, and Ms Pearcey and I discussed the similarities and differences:

- *How would you compare the two?*
- Both have the saxophones at the start, which is good. I suppose the difference is that I've got all of Simon's together and then I've got John's, but it's possible that I gave you slightly different availability times in the information I gave you. The afternoon is pretty much the same, but Ananya Tare had an issue with the original time slot so I moved her earlier.
- *Why is your schedule not sorted by grade in the last session?*
- It is actually grouped by teacher. I grouped just these exams by the teachers because they often come to support students beforehand. Not all teachers do this, but if you knew that they wanted to do this, that's just another thing to consider. But that is pretty insignificant, it's partially in that order because of the way my spreadsheet is organised.
- *Do you think it would be good to introduce some more leniency with the accompanist availability, say 5 minutes extra each end?*
- That would be an idea, like a contingency to promote grouping the accompanists. And you could highlight the exam and say 'this exam is slightly earlier/later' than their availability.
- *I guess allowing the schedule to be edited manually would solve that problem though?*
- Yeah, if I could just edit the schedule manually by dragging and dropping exams around, I would have full control over it, so any issues could be fixed.
- *Would you say that my schedule is better or worse than the one you created?*
- I think the only thing that I think is worse is that little bit there where John's is the middle of Simon's. But that might be down to the availability I told you he had. Sometimes you have to go back and renegotiate with accompanists, asking them to stay later. Apart from that, the schedule is great.

### *General scheduling questions*

Here are some questions and answers from the interview that are relevant to the scheduling of exams. Some answers have been summarised. These questions were asked once Ms Pearcey had seen all schedules, so she was well-informed about the system.

- *Do the schedules adhere to the TCL exam board regulations?*
- Yes, definitely.
- *Do you think it could have saved any more time or been more efficient?*
- I don't think so, no.
- *Would the accompanists be happy with the schedules produced?*

- Essentially, yes. They occasionally have to hang around for a bit longer, but that is sometimes a good thing, it gives them a break. Most accompanists are grouped very well. The odd exam is out of place.
- *Do you think the exams could have been grouped any better?*
- It is great, there is potentially more grouping than necessary, which is not a bad thing. If they were more mixed up it wouldn't matter. However, it is easier for examiners if exams are sorted by grades, so they would prefer this schedule. The odd exam is out of place, but that's just an exam here and there, not much to worry about.
- *Would you change anything about the scheduling functionality?*
- I don't think so. Maybe in the future, if you developed this commercially, you could give the option to manually swap students. After making the schedule, if somebody tells me they can't play at a certain time, it would be ideal if I could go into this nice display and just make manual changes. (she is referring to my edit function, which was removed in section 2.5.1). But this would be an extra thing, I think you've done a good job with developing what needs to be developed.
- *(I also showed her the error message displayed when accompanists have insufficient availability) What do you think about the error returned when accompanists have insufficient availability?*
- That's, again, very good. There would potentially be one thing that could make it a bit more useful. When it says who hasn't been scheduled, it would be good to know how much total exam time is needed to schedule everyone. That would help me when thinking about booking another day or half-day, because I would know how much is left to schedule.

### *Displaying specific information on the dashboard*

Ms Pearcey was quite pleased with the feature of displaying specific information on each user's dashboard. I showed her a view of the dashboard for an accompanist, a teacher, an accompanist **and** a teacher, and a student. She commented that the data displayed on screen was quite useful, as "it even says who the accompanist is". She also said the feature as a whole seemed quite sensible. I asked some further questions about this feature:

- *Is the schedule produced accurate and relevant to each supervisor?*
- Yes, definitely.
- *Is the schedule produced accurate and relevant to each student?*
- Yeah, it's perfect.
- *Would you prefer it if everyone could see the whole schedule, or just the exams they are involved in?*
- I think it's better to see what they need to see. So the current method is good.

### *Login functionality*

- *Are you happy that only you can access the scheduling functionality and other admin features?*
- Yes, I am. It is good for security as it prevents anyone else from messing with the schedule.

### *Usability features*

I also asked Ms Pearcey about the usability of the system, with the questions designed in section [2.7](#). I have summarised her feedback about the software's usability later on (in section [4.4](#)).

## 4.2 Functionality testing

The tables shown below contain the tests I created during section [2.7](#). As some functions have been developed quite differently compared to how they were designed, I have changed and added some tests. These changes are shown in **blue text**.

### (L) Login and registration

L	Description	Evidence	Successful?
1	Students can log in correctly, taken to the dashboard	login_tests.mp4 00:53 - 01:14	Y
2	Supervisors can log in correctly, taken to the dashboard	login_tests.mp4 01:52 - 02:06	Y
3	Admin can log in correctly and is taken to the dashboard	login_tests.mp4 02:09 - 02:21	Y
4	Students can register themselves correctly, and are redirected to the login page <b>with a message “Registration successful!”</b>	login_tests.mp4 00:00 - 00:53	Y
5	Supervisors can register themselves correctly, and are redirected to the login page <b>with a message “Registration successful!”</b>	login_tests.mp4 01:16 - 01:52	Y
6	Clicking logout will log the user out, taking them back to the index page	login_tests.mp4 01:14 and 02:09	Y
7	Only the admin can access scheduling functionality, as all pages should redirect unauthorised users away.	testing_L7.mp4 (full video)  login_tests.mp4 02:21 - 02:32	Y
8	The dashboard includes a link to the admin dashboard for the admin only	login_tests.mp4 02:21 - 02:32	Y

### Annotated evidence for login tests

The videos login\_tests.mp4 and testing\_L7.mp4 show that all of these tests have been met. In login\_tests.mp4, I have registered a new student, ‘Oliver Simpson’, with all of the following valid data:

- First name: Oliver
- Last name: Simpson
- Email: oliver.simpson@example.com
- Parent email: simpson.parent@example.com
- Password and confirm password: ‘Password123’

Submitting this data takes the user to the login screen, and displays the correct message, as expected. L4 is complete. The same student is then logged into the system, with the same (correct) email and password. This redirects the user to the dashboard, as he is logged in, so L1 is successful. L6 is also successful as clicking ‘logout’ in the navigation bar logs the user out properly. The same process is carried out for a supervisor with the following valid data:

- First name: Sophie
- Last name: Gorman
- Email: sophie.gorman@example.com
- Password and confirm password: ‘Password123’

The same process shows that L5 and L2 are successful. The admin is then logged in, with the following valid login details:

- Email: admin@email.com
- Password: Admin123

The correct login details logs the admin into the system, taking them to dashboard.php. So test L3 has passed. Notice that there is a message on the admin’s dashboard, which was not seen on any other dashboards. This message includes a link to the admin dashboard, which admins have exclusive access to. I clicked this link to show that it works, so L8 is successful.

To test L7, the same part of this video shows that admins can access the admin dashboard. In the video testing\_L7.mp4, I have logged in as a supervisor and student, and tried to access the 3 pages which only admins should have access to: dashboard-admin.php, schedule.php, schedule-complete.php . Firstly, their dashboard has no link to any of these pages, to deter users from trying to access this functionality. Trying to enter the URL directly will cause them to be redirected back to the dashboard, so they are blocked from accessing these pages.

There are some more subtle aspects of this functionality. The start of the video shows that the dashboard redirects users to the login page if they have not been logged in. Once logged in, the navigation bar changes to show the actions that the user can take. Overall, any secure pages have clearly been shown to redirect unauthorised users away, so L7 is successful.

## (I) Gathering inputs for a new schedule

I	Description	Evidence	Successful?
1	<p>Start new schedule button will delete and create BasicInfo, Examinees, ExamAccomps and Schedule in the database (to prepare the database for a new schedule) and redirect to new-schedule.php</p> <p><b>UPDATE:</b> It should only empty BasicInfo and Schedule, not the other tables. It should redirect to schedule.php</p>	testing_I1.mp4 (full video)	Y

2	The exam's start date (valid data) can be entered and is stored in the database <b>when the 'Schedule' button is clicked.</b>	schedule_button.mp4 (full video)	Y
3	Valid accompanist availability can be entered and is stored in the database	N/A	N
4	A valid CSV file of examinee information can be entered and is stored in the database	N/A	N

testing\_I1.mp4 shows the full database, before clicking 'Start New Schedule'. With the admin login (Lorne Pearcey), I have gone to the admin dashboard, and clicked 'Start New Schedule'. This redirects me to schedule.php, as expected. Reloading the database now shows that BasicInfo and Schedule have been emptied, but Examinees and ExamAccomps are not affected. Therefore, I1 is successful.

After entering a valid start date in schedule\_button.mp4, I clicked the schedule button. The next page displays this same date, showing that it has been stored correctly in the database, and then retrieved.

Tests I3 and I4 have not been carried out, as accompanist and examinee inputs have not been developed, so there is nothing to test. I will talk about this in more depth in the evaluation.

## (S) Scheduling

The produced schedule takes many different inputs, and therefore each schedule will be different. During iterative testing, I comprehensively tested all possible inputs for each function. During post-development testing I believe it would be more appropriate to test my scheduling functionality against 3 sets of real data, which will give a representative sample to allow me to thoroughly test how the schedule performs in a real situation. Therefore, my column for the success is split to show how the test performed using each dataset.

S	Description	Evidence	Successful? (using 3 sets of real data)		
			Spring	Summer	Winter
1	The schedule function will produce a schedule that meets all TCL regulations	Figure 4.1a/b/c (see below)	Y	Y	Y
2	Breaks are scheduled within 10 minutes of the ideal break times set by Ms Pearcey	Figure 4.1a/b/c (see below)	Y/N	Y	Y
3	The scheduled wastes as little time as possible	Figure 4.1a/b/c (see below)	Y	Y	Y
4	The most urgent exams are attempted to	Figure 4.1a/b/c (see	Y	Y	Y

	be scheduled first	below)			
5	The schedule always adheres to accompanist availability	Figure 4.1a/b/c (see below)	Y	Y	Y
6	Exams are reasonably grouped by accompanist	Figure 4.1a/b/c (see below)	Y/N	Y/N	Y
7	Within each accompanist block, exams are grouped by instrument family	Figure 4.1a/b/c (see below)	Y	Y	Y
8	Within each instrument family, exams are grouped by instrument	Figure 4.1a/b/c (see below)	Y	Y	Y
9	Within each instrument, exams are sorted ascending by grade	Figure 4.1a/b/c (see below)	Y	Y	Y
10	The schedule is stored within the Schedule table in the database	Figure 4.2a	Y		
11	Given valid data, the 'Schedule' button runs the schedule function and redirects to schedule-complete.php	Schedule_button.mp4 00:15 - 00:30	Y		

Figure 4.1a, 4.1b, and 4.1c display the Spring, Summer, and Winter schedules produced by my software. They are shown above, but for simplicity I have repeated them above. For each dataset, I have shown why the tests have/have not been successful.

*Testing with Spring 2023 data*

## Exam Schedule | 1st Mar 2024

Student Name	Instrument	Grade	Accompanist Name	Start Time
Sam Nouhov	Jazz Saxophone	6	Elliot Drew	<b>09:00</b>
Daniel Okpla	Classical Guitar	1		<b>09:23</b>
Olivia Yang	Classical Guitar	2		<b>09:36</b>
Reena Koiri	Classical Guitar	2		<b>09:49</b>
Maryam Mohamed	Clarinet	5	John Meadows	<b>10:02</b>
Ella Goldberg	Clarinet	6	John Meadows	<b>10:20</b>

Charlotte Wood	Clarinet	7	John Meadows	<b>10:58</b>
Matteo Gianni	Baritone	6	Simon Palmer	<b>11:21</b>
Clover Webster	Cornet	5	Simon Palmer	<b>11:44</b>
Edward Cheung	French Horn	2	Simon Palmer	<b>12:02</b>
Basheir Said	Trumpet	1	Simon Palmer	<b>12:15</b>
<hr/>				
Lemuel Adjei	Trumpet	3	Simon Palmer	<b>13:28</b>
Kirsty Jenkins	Clarinet	7	John Meadows	<b>13:41</b>

Benjamin Law	Trumpet	6	Simon Palmer	<b>14:04</b>
Shreya Jondhale	Violin	5	Simon Palmer	<b>14:27</b>
William Lin	Violin	5	Simon Palmer	<b>14:45</b>
<hr/>				
Anupama Harish	Violin	6	Simon Palmer	<b>15:18</b>
Anushtup Chatterjee	Violin	6	Simon Palmer	<b>15:41</b>
Ananya Tare	Classical Guitar	3		<b>16:04</b>
Alicia Dieu	Classical Guitar	3		<b>16:17</b>
Levi Dandy	Classical Guitar	3		<b>16:30</b>
Sana Khan	Classical Guitar	3		<b>16:43</b>

(repeated) Figure 4.1a: Schedule produced using Spring 2023 data

In order to meet all TCL regulations, the schedule must follow these regulations:

- Each session is max. 2 hours
- Lunch break must be scheduled before 12:45
- Total exam time is max. 6.5 hours
- Exams end at 5pm at the latest
- Lunch break is 1 hour, 2 others breaks are 15 minutes

These regulations have been explained and substantiated clearly in sections [1.1.2](#) and [1.1.3](#).

The first session runs from 09:00 to 10:43, the second from 10:58 to 12:28, the third from 13:28 to 15:03, and the fourth from 15:18 to 16:56. None of these sessions last longer than 2 hours. The lunch break is 1 hour, 12:28 - 13:28. The other 2 breaks are exactly 15 minutes. The last exam ends at 16:56. All exams in total last 6 hours and 26 minutes. These values are all within the regulations, so the TCL regulations have been met, and Test S1 is successful.

The ideal times for breaks were set by Ms Pearcey. These times are: 10:45, 12:30, 15:15. My schedule has set breaks to start at 10:43, 12:28, and 15:03. The first 2 breaks are excellent, as they have been scheduled within 2 minutes of this time. The final break, however, is 12 minutes before the ideal start time. Test S2 is therefore partially successful for this dataset.

The schedule has wasted as little time as possible, as the final schedule is within 4 minutes of the latest possible time! From 09:00 to 16:56, only exams and mandatory breaks have been scheduled, nothing else. There would be no way to save any more time, and Ms Pearcey felt the same way in her interview, saying 'I don't think so' when asked if the schedule could have saved any more time.

The most urgent exams have been attempted to be scheduled first, as the schedule has tried to schedule exams with accompanists first. On top of this, it has tried to schedule accompanists with the least free time first. This is why Elliot Drew's exam is first, and then John Meadows' exams are scheduled as soon as he becomes available. The schedule always adheres to accompanist availability as well. The data provided says that Simon Palmer is available from 11:00-17:00, Elliot Drew is available from 09:00- 11:30, and John Meadows is available from 10:00 to 15:30. The schedule clearly shows that all exams stick to this availability.

When it comes to grouping exams by accompanist, the schedule has mostly done very well. However, as Ms Pearcey pointed out, the 13:41 exam is out of place, as John Meadows has an exam scheduled over 2 hours after all of his other exams. Apart from this one exam, the groupings are good.

For tests S7-9, the schedule clearly groups exams correctly. Simon Palmer's brass exams are grouped separately from his strings exams. Whenever possible, the same instruments are grouped together and sorted ascending by grade. The classical guitar exams show that exams are sorted by grade very accurately, this is true across all datasets.

The final 2 tests (S10-11) have not been split by the data used. This is because their function does not depend on the intricacies of the dataset, so the test does not need to be repeated with multiple sets of data.

ScheduleID	ExamineeID	StartTime
1	42	09:00:00
2	9	09:23:00
3	40	09:36:00
4	41	09:49:00
5	38	10:02:00
6	34	10:20:00
7	31	10:58:00
8	39	11:21:00
9	32	11:44:00
10	33	12:02:00
11	29	12:15:00
12	36	13:28:00
13	35	13:41:00
14	30	14:04:00
15	43	14:27:00
16	44	14:45:00
17	27	15:18:00
18	28	15:41:00
19	6	16:04:00
20	26	16:17:00
21	37	16:30:00
22	45	16:43:00

Figure 4.2a: Schedule table in the database

This screenshots shows that the schedule is stored correctly in the database, all 22 examinees have a record in the table. To further prove that this schedule is stored in the database, Figure 4.1a/b/c all retrieve data from this database to display the schedule! Proof for test S11 is shown in the video schedule\_button.mp4. Proof that the schedule function has run is shown as the schedule has been displayed, displaying the exams that were scheduled by this function. Furthermore, the schedule function returns a message “All examinees have been scheduled” in the variable \$alert. This message is displayed at the top of schedule-complete.php at 00:19. Tests S10 and S11 are therefore successful.

*Testing with Summer 2023 data*

**Exam Schedule | 1st Jul 2024**

Student Name	Instrument	Grade	Accompanist Name	Start Time
Marriam Javed	Trombone	1	Simon Palmer	<b>09:00</b>
Deeksha Sharma	Trombone	4	Simon Palmer	<b>09:13</b>
Aman Koiri	Clarinet	5	John Meadows	<b>09:31</b>
Clara Hilton-Widdows	Clarinet	5	John Meadows	<b>09:49</b>
Rinsola Alatise	Clarinet	5	John Meadows	<b>10:07</b>
Samia Islam	Clarinet	5	John Meadows	<b>10:25</b>
<hr/>				
Inaaya Kassim	Violin	1	Simon Palmer	<b>10:58</b>
Birle Tenekeci	Violin	2	Simon Palmer	<b>11:11</b>
Louisa Dilling	Violin	6	Simon Palmer	<b>11:24</b>
Tamsin Howard	Violin	7	Simon Palmer	<b>11:47</b>
Diya Sharath	Classical Guitar	1		<b>12:10</b>
Lucas Evans	Classical Guitar	1		<b>12:23</b>
<hr/>				
Catherine Mae Villabroza	Jazz Saxophone	7	Elliot Drew	<b>13:36</b>
<hr/>				
Ryheem Miah	Saxophone	1	Elliot Drew	<b>13:59</b>
Mikayla Rodney	Saxophone	3	Elliot Drew	<b>14:12</b>
Mithun Kesavan	Classical Guitar	1		<b>14:25</b>
Sara Sadiq	Classical Guitar	1		<b>14:38</b>
Zubair Ahmed	Classical Guitar	1		<b>14:51</b>
Zaynah Jamal	Classical Guitar	2		<b>15:04</b>
<hr/>				
Aakshat Kumar	Classical Guitar	3		<b>15:32</b>
Chardé Levermore	Classical Guitar	3		<b>15:45</b>
Sana Khan	Classical Guitar	4		<b>15:58</b>
Anesha Mitra	Classical Guitar	5		<b>16:16</b>
Elizabeth Shpectorov	Classical Guitar	7		<b>16:34</b>

(repeated) Figure 4.1b: Schedule produced using Summer 2023 data

Tests S1-9 have been repeated on the Summer 2023 dataset, which includes some natural variations and hence enables the testing to be more realistic. All TCL regulations are being met by this schedule. No session is longer than 2 hours, a 1 hour lunch break is scheduled at 12:36 (which is before 12:45), and two 15 minute breaks have also been scheduled. All exams add up to 6 hours and 27 minutes, and the final exam ends at 16:57 (3 minutes below the limit). Therefore, test S1 is successful. The 3 breaks are scheduled at: 10:43, 12:36, and 15:17. Respectively, these breaks are 2, 6, and 2 minutes from the ideal time, which shows that test S2 is successful.

Once again, the schedule has wasted minimal to no time. All examinees have been scheduled, and the only time where exams are not scheduled is due to mandatory breaks. The availability of accompanists for this dataset is:

- Simon Palmer: 09:00 - 17:00
- John Meadows 09:15 - 14:15
- Elliot Drew 12:45 - 17:00

The accompanist availability has been met by this schedule, and the accompanists with most priority are scheduled as soon as they are available. This is shown as John Meadows, who has many exams but limited availability, is scheduled from the first possible time slot that meets his availability. Therefore, tests S4-5 are successful.

Once again, the grouping of accompanists is good. Ms Pearcey said the schedule produced was ‘ideal’. However, she suggested that it would be more logical to schedule all of Simon Palmer’s exams in the second session, as it would save his time. This one small change is the only issue Ms Pearcey had with this schedule, and is the only problem regarding the accompanist grouping. Mostly, accompanists are grouped quite well. S6 is therefore partially successful

The schedule clearly shows all exams grouped into instrument families (within each accompanist block), and then grouped by instrument and sorted by grade. This is very accurate, and Ms Pearcey believed so too during our interview.

*Testing with Winter 2023 data*

**Exam Schedule | 24th Jan 2024**

<b>Student Name</b>	<b>Instrument</b>	<b>Grade</b>	<b>Accompanist Name</b>	<b>Start Time</b>
Doyinsola Oliyide	Saxophone	1	Elliot Drew	<b>09:00</b>
Akshay Suglani	Saxophone	3	Elliot Drew	<b>09:13</b>
Zaid Rassam	Trumpet	1	Simon Palmer	<b>09:26</b>
Jiayi (Eva) Chen	Clarinet	3	John Meadows	<b>09:39</b>
Henna Naveed	Clarinet	5	John Meadows	<b>09:52</b>
Leya Rahman	Trumpet	5	Simon Palmer	<b>10:10</b>
Xander Davis	Trumpet	6	Simon Palmer	<b>10:28</b>
Lily-Marie Le Blanc	Viola	3	Simon Palmer	<b>11:06</b>
Milaura Fernando	Violin	1	Simon Palmer	<b>11:19</b>
Maryam Rahman	Violin	5	Simon Palmer	<b>11:32</b>
Aliza Rayhan	Flute	6	Simon Palmer	<b>11:50</b>
Lauren Pumphrey	Flute	6	Simon Palmer	<b>12:13</b>
Ahana Seeam	Classical Guitar	1		<b>13:36</b>

Aisha Nashmia	Classical Guitar	1		<b>13:49</b>
Aroush Haider	Classical Guitar	1		<b>14:02</b>
Eshal Aamir	Classical Guitar	1		<b>14:15</b>
Farishta Dosanjh	Classical Guitar	1		<b>14:28</b>
Shailey Sivathasan	Classical Guitar	1		<b>14:41</b>
Daniel Okpla	Classical Guitar	2		<b>14:54</b>
Anisha Kawle	Classical Guitar	3		<b>15:07</b>
Sri Grandhi	Classical Guitar	3		<b>15:35</b>
Aakshat Kumar	Classical Guitar	4		<b>15:48</b>
Ananya Tare	Classical Guitar	4		<b>16:06</b>
Nikhil Gilliam	Classical Guitar	4		<b>16:24</b>
Tula Hobbs	Classical Guitar	5		<b>16:42</b>

(repeated) Figure 4.1c: Schedule produced using Winter 2023 data

All tests are successful for this data. For the same reasons shown in previous schedules, the TCL regulations are all being met. The only difference is that the total exam time is exactly 6.5 hours, and the schedule ends exactly at 5pm. This has taken the schedule right to the limit of its validity, so has incidentally become a successful result using boundary data. S3 is successful, as all examinees have been scheduled even though the maximum allowed duration of exams has been reached. There would be no way to make this schedule any more efficient than it is currently, it would always end at 5pm or later. The breaks are scheduled 6,6, and 5 minutes from the ideal time, which is enough to pass test S2.

Accompanist availability is shown below:

- Simon Palmer 09:00 - 17:00
- Elliot Drew 09:00 - 11:30
- John Meadows 09:30 - 15:30

The most urgent exams have been scheduled first, as all accompanists are scheduled in the morning. Although both Simon Palmer and Elliot Drew are available at 09:00, Elliot is scheduled first. This is because he has much less free time (as Mr Palmer is available for the whole day), so he is prioritised due to his urgency. This logic is carried out throughout the whole schedule, so S6 is successful.

Tests S7-9 are all successful. A good example of grouping by instrument can be seen during Simon Palmer's exams, with viola and violin exams being grouped together. The schedule shows clearly that exams are grouped by instrument family, instrument and then sorted by grade.

## (P) Publishing

P	Description	Evidence	Successful?
1	'View Existing Schedule' button takes user to schedule-complete.php and displays the full schedule	publishing.mp4 01:00-01:20	Y
2	'Publish' button will set IsPublished to 1 and redirect the user to the <b>dashboard admin dashboard</b> with a message "The schedule has been published!"	publishing.mp4 01:17-01:24  Figure 4.2b	Y
3	If nothing is published, users will receive a message to acknowledge this on their dashboard.	publishing.mp4 00:18-00:25 00:35-00:40	Y
4	If nothing is published, students will receive an additional message regarding the communication of their results	publishing.mp4 00:18-00:25	Y
5	Once published, all students view specific information about their upcoming exam	publishing.mp4 01:47-01:55	Y
6	Once published, all supervisors view specific schedule information about upcoming exams	publishing.mp4 02:10-02:15	Y

	they are accompanying	02:28-02:38 02:50-03:05	
7	Once published, all supervisors view specific schedule information about upcoming examinees they are teaching.	publishing.mp4 02:10-02:15 02:28-02:38 02:50-03:05	Y
8	Each user's timetable abstracts any unnecessary data from view	publishing.mp4 01:47-01:55 02:10-02:15 02:28-02:38 02:50-03:05	Y
9	All users will see a message to contact Ms Pearcey if there are any concerns, with her email included in this message.	Publishing.mp4 (seen throughout the video, one example is 00:22)	Y
10	The admin is able to download a PDF of the schedule by clicking the 'Download PDF' button	N/A	N
11	The PDF contains the full schedule, with the following fields displayed: Examinee name, Instrument, Grade, Accompanist name, Start time	N/A	N
12	schedule-complete.php displays the schedule split by breaks.	publishing.mp4 01:00-01:20	Y
13	Once published, the 'Schedule Again' and 'Publish' buttons disappear from the screen.	publishing.mp4 01:27-01:33	Y
14	The timetable shown for teachers includes an additional column 'Teacher name'.	Publishing.mp4 02:28-02:38, 02:50-03:00	Y

The video publishing.mp4 clearly shows why these tests are successful.

Start Date	Timestamp	Is Published
2024-06-26	2024-01-23 23:03:03	1

Figure 4.2b: The publish button changes IsPublished to 1 in the table 'BasicInfo'

For test P2, the above screenshot shows the table 'BasicInfo' after clicking the publish button. IsPublished is now set to 1, which causes timetables to be displayed on every user dashboard. For test P6-7, I logged in with several different supervisors. I logged in Simon Palmer, who is just an accompanist, John Meadows, who is both an accompanist and

teacher, and John Phillips, who is just a teacher. There is clearly no unnecessary data on the screen, all data displayed on the dashboard is relevant to the user. Ms Pearcey confirmed that all data displayed is relevant to the user during our interview:

- *Is the schedule produced accurate and relevant to each supervisor?*
- Yes, definitely.
  
- *Is the schedule produced accurate and relevant to each student?*
- Yeah, it's perfect.
  
- *Is there any unnecessary information displayed on the screen when the schedules are displayed?*
- No.

Ms Pearcey's comments confirm that P8 is successful. P10 and P11 have not been tested, as I have not developed the feature of producing a PDF. This limitation of the solution will be discussed in the evaluation.

## 4.3 (R) Robustness testing

Data used in these tests is invalid data

R	Description	Input data	Evidence	Successful?
1	Empty fields are rejected during registration	NULL	robustness_registration.mp4 00:10-00:14	Y
2	Invalid name is rejected during registration	'rohan1'	robustness_registration.mp4 00:35-00:42	Y
3	Invalid email is rejected during registration	'a@b'	robustness_registration.mp4 01:00-01:10	Y
4	Invalid password is rejected during registration	'password'	invalid_password.mp4 (full video)	Y
5	A registered email is rejected during student registration	'aakshat.kumar@example.com'	robustness_registration.mp4 01:10-01:45	Y
6	Unregistered (invalid) email is rejected during login	'rohan@example.com'	robustness_registration.mp4 02:20-02:30	Y
7	Invalid password is rejected during login	Email: 'aakshat.kumar@example.com'  Password: 'IncorrectPass1'	robustness_registration.mp4 02:35-02:46	Y
8	If the date is in the past, it is rejected.	Start Date: 01/01/2024	Invalid_date.mp4 (full video)	Y
9	There is a drop down list that only displays registered supervisors, so only they can be selected as accompanists	N/A	N/A	N
10	If one user is already the school accompanist, selecting the school accompanist for another user will return an error message.	N/A	N/A	N
11	If the examinee file is not a CSV, it is rejected.	N/A	N/A	N
12	When entering examinee info, if each row does not contain 5 elements with correct data types, the file is rejected.	N/A	N/A	N
13	When entering examinee info, if an	N/A	N/A	N

	element is not verified as a valid foreign key in the database, the entire file is rejected.			
14	The schedule is constantly validated to be meeting the regulations, and if an error is found, the program stops and displays it.		Figure 4.3a/b/c/d	Y
15	When too many examinees are entered into the database, the day is ended before 5pm to ensure a valid schedule is produced		Figure 4.3c/d	Y
16	Only once regulations are broken will the schedule end, in order to schedule as many exams as allowed.		Figure 4.3b/d	Y/N
17	When insufficient availability is entered, the function will recognise this and display an error message.		Figure 4.3a	Y
18	The error message will display the specific names of accompanists with insufficient availability		Figure 4.3a	Y
19	A registered email is rejected during <u>supervisor registration</u>	'simon.palmer@example.com	robustness_registration.mp4 01:55-02:10	Y
20	If the Examinees table is empty, an error message is displayed and the schedule function will not run (when the schedule button is clicked)	Examinees table is empty	empty_examinees.mp4 (full video)	Y
21	If examinees are not scheduled, their names are displayed on the screen		Figure 4.3a/c	Y

The videos provide evidence to show that these tests have been successful. For test R4, please ignore the pop-up message displayed at 00:30, this is a result of Google Chrome's security features, and is not part of my software. The input data provided for this table mostly shows the piece of invalid data that is causing the error, all other data entered is valid. The videos show that the error messages displayed are very specific and clear, indicating specifically what the issue is.

Tests R9-13 have not been carried out, and therefore the tests have failed. This is because I have not developed the feature to input examinee or accompanist information, due to the time constraints of this project. This limitation will be addressed during my evaluation.

## Scheduling Results

The following examinees have not been scheduled: Henry Reed , Carter Evans , Michael Brown , Madison Barnes , Logan Wilson , Aiden Fisher , Jackson Ward , Isabella Scott , Lucas Ross , Sophie Hill , Ava Moore , Avery Cooper , Liam White , Grace Kelly

ERROR:

Sarah Johnson has too many exams / is not available enough. They need at least 18 minutes more.

Figure 4.3a: Messages due to insufficient accompanist availability

The blue message shows clearly which examinees have not been scheduled, including their full names in the text. The red error message names the accompanist with insufficient availability, Sarah Johnson. It states the issue clearly, and provides an estimated amount of additional availability needed. R17,18, and 21 are therefore successful.

### Exam Schedule | 28th Jan 2024

Student Name	Instrument	Grade	Accompanist Name	Start Time
Emma Davis	French Horn	1	David Williams	09:00
Emily Miller	Classical Guitar	3	David Williams	09:13
Joe Davies	Classical Guitar	4	David Williams	09:26
Alex Johnson	Classical Guitar	6	David Williams	09:44
Lily Morris	Clarinet	1	David Williams	10:07
Ethan Clark	Trumpet	1		10:20
Mia Baker	Trumpet	5		10:33
<hr/>				
Sophia Hall	Trombone	2	Sarah Johnson	11:06
Daniel Williams	Trombone	5	Sarah Johnson	11:19
Noah Turner	Trombone	5	Sarah Johnson	11:37
Olivia Jones	Trumpet	8	Sarah Johnson	11:55
Jane Smith	Violin	2	Sarah Johnson	12:23
<hr/>				
Chloe Harrison	Jazz Saxophone	1	Christopher Smith	13:36

Figure 4.3b: Schedule produced from insufficient accompanist availability

Each time something is scheduled, the algorithm has validated the status of this schedule against regulations. This is shown by the appropriate scheduling of breaks, as the regulations would not be met if an exam was scheduled, causing the break to be scheduled. Also, the schedule has stopped scheduling as soon as it detected Sarah Johnson has insufficient availability. It has not scheduled all possible exams, going until she becomes

unavailable. Instead, it stops the schedule as soon as an error is found. While this is sensible, it may be more useful and informative to schedule all possible exams, and stop scheduling once she can no longer accompany any more. It is clear that not all exams have been scheduled as possible.

The screenshot shows a software application window titled "SoundBro." at the top left. To its right are two buttons: "Dashboard" with a bar chart icon and "Logout" with a user profile icon. The main content area has a light blue header with the title "Scheduling Results" in bold blue text. Below this, a message in a white box states: "The day has ended. The following examinees have not been scheduled: Ava Moore , Avery Cooper , Liam White , Grace Kelly". Underneath this message is a section titled "Exam Schedule | 3rd Feb 2024" in bold black text. A table below this section displays student information:

Student Name	Instrument	Grade	Accompanist N
Sophia Hall	Trombone	2	Sarah Johnson

Figure 4.3c: Messages displayed when too many examinees are entered

R21 and R13 are successful as the message clearly displays the names of unscheduled examinees. In this case, too many examinees have been entered into the system, so it has ended the day before scheduling everyone. Therefore, it displays the message 'the day has ended' to indicate this. This contributes to the success of R15

## Exam Schedule | 3rd Feb 2024

Student Name	Instrument	Grade	Accompanist Name	Start Time
Sophia Hall	Trombone	2	Sarah Johnson	<b>09:00</b>
Daniel Williams	Trombone	5	Sarah Johnson	<b>09:13</b>
Noah Turner	Trombone	5	Sarah Johnson	<b>09:31</b>
Olivia Jones	Trumpet	8	Sarah Johnson	<b>09:49</b>
Jane Smith	Violin	2	Sarah Johnson	<b>10:17</b>
Henry Reed	Clarinet	5	Sarah Johnson	<b>10:30</b>
<hr/>				
Chloe Harrison	Jazz Saxophone	1	Christopher Smith	<b>11:03</b>
Carter Evans	Jazz Saxophone	5	Christopher Smith	<b>11:16</b>
Emma Davis	French Horn	1	David Williams	<b>11:34</b>
Emily Miller	Classical Guitar	3	David Williams	<b>11:47</b>
Joe Davies	Classical Guitar	4	David Williams	<b>12:00</b>
<hr/>				
Alex Johnson	Classical Guitar	6	David Williams	<b>13:18</b>
Lily Morris	Clarinet	1	David Williams	<b>13:41</b>
Ethan Clark	Trumpet	1		<b>13:54</b>
Mia Baker	Trumpet	5		<b>14:07</b>
Michael Brown	Trumpet	7		<b>14:25</b>
Madison Barnes	Classical Guitar	1		<b>14:48</b>
Logan Wilson	Classical Guitar	3		<b>15:01</b>
<hr/>				
Aiden Fisher	Classical Guitar	3		<b>15:29</b>
Jackson Ward	Classical Guitar	4		<b>15:42</b>
Isabella Scott	Classical Guitar	7		<b>16:00</b>
Lucas Ross	Viola	3		<b>16:23</b>
Sophie Hill	Violin	1		<b>16:36</b>

Figure 4.3d: Schedule produced when too many examinees are entered

This schedule shows that validation has occurred throughout the scheduling process, as the program stops scheduling and displays a message as soon as an exam is breaking the regulations. In this case, it has found that the next exam is ending after 5pm, which is invalid,

so a message is returned (figure 4.3c) to say ‘the day has ended’. In this example, as many exams have been scheduled as possible, as there is no way to schedule any more exams without breaking the regulations in some way. R16 is therefore partially successful, due to success in figure 4.3d, but failure in figure 4.3b.

## *4.4 Usability testing*

To test the usability of the system, I asked my stakeholders the following questions (as designed in [2.7](#)):

1. Do you think this interface is simple and intuitive?
2. Are the scheduling results easy to understand?
3. Would you want any other data to be displayed?
4. Is the on-screen help useful?
5. Is there any unnecessary information on screen at any point in the system?
6. Would you change anything to make the system more usable?

I asked these further questions to Ms Pearcey:

7. Is the ‘schedule’ button easy to press?
8. Is the ‘publish’ button easy to press?
9. Considering people with varying age and technological dexterity will use this system, would you say that the interface is easy to understand?

### *Ms Pearcey usability feedback*

The following answers are taken from my interview with Ms Pearcey conducted at the start of post-development testing. The full transcript can be found in the [Appendix](#). Some answers have been summarised for clarity.

1. Yes, I do. It's very, very well made.
2. Yes. Very easy.
3. Not that I can think of. The teacher's name column is a good idea.
4. Yes, they are quite helpful. I don't think I would want any more messages.
5. No.
6. The only thing that might be nice would be to save the previous schedule somewhere so that if you did accidentally delete it, you could see it or you could compare schedules potentially.
7. It's very easy and clear. Easy to press. Yeah.
8. Yeah. It's very clear.
9. Yeah.

### *Cyrus usability feedback*

I showed my final solution to Cyrus and Darius, who gave their opinion on the usability:

1. Yes, it looks quite professional and well-suited to the system. I found it quite easy to go to different pages.
2. Yes. It's very easy to see when your exam is. It is a tiny bit confusing when the title of the table is shown, but it says “you haven’t been scheduled” or something.
3. No
4. It is very useful, any more text and it might get a bit overpowering. You could consider having that little question mark icon with an explanation inside.
5. Not really

6. I think it's quite usable already, the icons are a nice touch. I'm not sure how difficult this would be, but maybe you could include some accessibility features to make it usable for everyone.

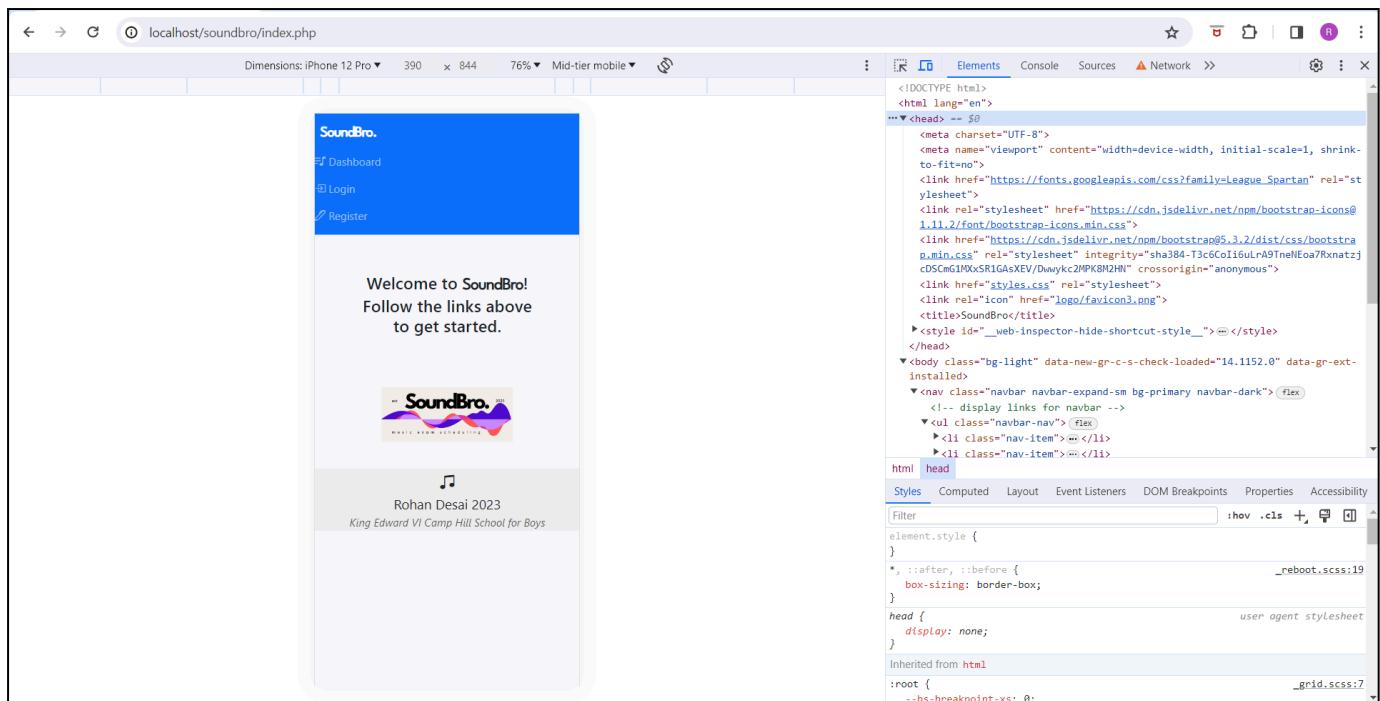
### *Darius usability feedback*

1. Yes. It looks very similar to the interface we designed, which is great. It feels quite modern.
2. Yes, very easy.
3. It's fine as it is. I wouldn't change it.
4. Yes. It's very direct and honest.
5. No
6. The footer you have on every page should be at the bottom of the screen, not halfway up. It would make it look a lot more professional. I was thinking that you could use that 'sign in with google' thing instead of making logins, because that way you wont forget this password. I don't think a year 7 would remember a password that he uses once, maybe twice a year. That's just an idea though, overall it is very easy to use.

Overall, my stakeholders are quite happy with the usability of the system. Ms Pearcey seemed very pleased with the interface, which is especially good as she will use this system the most. She mentioned that the 'schedule' and 'publish' buttons were easy to press and very clear, showing that this designed usability feature (from section [2.3](#)) has now been developed successfully. She made the suggestion that it would be better if previous schedules could be stored somewhere, and then loaded. This would save time by loading an old schedule instead of starting a new one. Cyrus and Darius also liked the interface, saying it looks 'professional' and 'modern'. They made some suggestions to improve usability even more, by changing the how onscreen help is provided, including accessibility features, and incorporating Google OAuth ('sign in with google' mentioned by Darius). These features would improve usability, but all users were satisfied with the system and interface overall. They all agreed that the onscreen help is useful, all necessary data is displayed, and it's easy to understand the timetable displayed on the dashboard.

### *Mobile compatibility*

In section 2.3, I mentioned that an important usability feature is to make the software compatible with mobile devices, as well as desktops. Therefore, I used Bootstrap 5, a HTML/CSS framework with lots of support for developing mobile-first websites. By styling most of the interface using Bootstrap, I have tried to achieve this usability feature during all stages of development. In order to test this, I used the 'inspect element' feature on google chrome and simulated a mobile screen on my laptop. The size of the simulated screen is based on the dimensions of the iPhone 12 Pro screen.



Simulating a mobile device accessing the website.

**SoundBro.**

- Dashboard
- Login
- Register

Welcome to SoundBro!  
Follow the links above to get started.



Rohan Desai 2023  
King Edward VI Camp Hill School for Boys

**SoundBro.**

- Dashboard
- Login
- Register

## Login

Email address

Password

**Login**

Don't have an account? [Register here](#)

Rohan Desai 2023  
King Edward VI Camp Hill School for Boys

## Register a Student

First name

Names must be 2-50 characters, with no numbers or special characters (except hyphens).

Last name

Email address

Parent email address

We'll only send you and your parent an email when the schedule is created.

Password

Password must be at least 8 characters long and include at least one uppercase letter, one lowercase letter and one number.

Confirm Password

**Register**

If you have an account, login [here](#)

**SoundBro.**

- Dashboard
- Logout

## Dashboard

Welcome, Aakshat!

Nothing has been scheduled yet. Check again later.  
If you have recently taken an exam, results will be displayed on the music notice board, or you will be contacted by your music teacher.

Email Ms. Pearcey at [admin@email.com](mailto:admin@email.com) if you have any concerns.

Rohan Desai 2023  
King Edward VI Camp Hill School for Boys

**SoundBro.**

- Dashboard
- Logout

## Dashboard

Welcome, Aakshat!

### Exam Schedule | 26th Jan 2024

Student Name	Instrument	Grade	Accompanist Name	Start Time
Aakshat Kumar	Classical Guitar	4		15:48

Email Ms. Pearcey at [admin@email.com](mailto:admin@email.com) if you have any concerns.

Rohan Desai 2023  
King Edward VI Camp Hill School for Boys

**SoundBro.**

- Dashboard
- Logout

## Dashboard

Welcome, Aakshat!

### Exam Schedule | 26th Jan 2024

Student Name	Instrument	Grade	Accompanist Name	Start Time
Aakshat Kumar	Classical Guitar	4		15:48

Email Ms. Pearcey at [admin@email.com](mailto:admin@email.com) if you have any concerns.

Rohan Desai 2023  
King Edward VI Camp Hill School for Boys

The screenshot shows the SoundBro website's dashboard for a user named Simon. The top navigation bar includes links for 'Dashboard' and 'Logout'. The main content area is titled 'Dashboard' and displays a welcome message 'Welcome, Simon!'. Below this, there are two sections: 'Exam Schedule | 26th Jan 2024 | Teacher Info' and 'Exam Schedule | 26th Jan 2024 | Accompanist Info'. The 'Teacher Info' section contains a table with columns for Student Name, Instrument, Grade, Teacher Name, and Accompanist Name. A message states 'You aren't teaching any upcoming examinees.' The 'Accompanist Info' section contains a similar table with data for three students: Zaid Rassam, Leya Rahman, and Xander.

Student Name	Instrument	Grade	Teacher Name	Accompanist Name
Zaid Rassam	Trumpet	1	Simon Palmer	
Leya Rahman	Trumpet	5	Simon Palmer	
Xander	Trumpet	6	Simon Palmer	

The images above show some commonly used screens of the website. Overall, the website has adjusted quite well to the dimensions of the screen. However, the navigation bar now takes up too much space on the screen. The containers for logging in and registering users are clearly visible and have adapted well to this change. None of the functionality of the website has been affected. The dashboard for a student (Aakshit Kumar) is shown in 3 screens. The first has displayed the relevant information on the dashboard very well, when nothing is published. However, the timetable has not been displayed on the screen in a very professional manner. The styling is still consistent, and the website is still easy to use and navigate, but this table is too wide for the screen size, and there is some overflow. This is shown in the third image of Aakshit's dashboard. If he wants to see his start time, he must scroll the page horizontally to access it, and all other parts of the website end while the table is displayed. This looks quite unpolished, and hides the most important information from direct view. Apart from this, the screens generally are quite easy to use. Simon Palmer's dashboard has also been shown above. Compared to the desktop screen dimensions, the only issue with this page is (once again) that the start time of each exam cannot be seen immediately. This can potentially be confusing for users, and make the system more difficult to use.

I believe that the usability features designed and developed have mostly been successful. The buttons are clear and emphasised, and easy to press. The onscreen help is accurate and useful. The interface is clean, simple and consistent throughout all pages. The use of icons has further reinforced this professional look, and colour coordination is used to emphasise alerts and error messages. The name 'SoundBro' and its logo has been praised by all of my stakeholders, showing that it has left a positive impression on the user. There are several possible improvements to be made to the usability of the system, as suggested

by my stakeholders. However, I believe that the main issue with this website's usability comes from the compatibility issues with mobile devices, as the fundamental start times of each exam are not directly visible.

# 5 Evaluation

## 5.1 Meeting the success criteria

The following table has cross-referenced all success criteria from stage [1.7](#) with the post-development testing from stage [4](#). Evidence for each test is shown clearly and annotated in stage 4, along with the test's success. If the test is successful, the test reference is highlighted in green (eg. [R2](#)). If unsuccessful, it is highlighted in red (eg. [R12](#)). If partially successful, it is highlighted in orange (eg. [S6](#)).

For each success criteria, all relevant tests have been considered when evaluating if the criteria has been met or not. The outcome of each success criteria is clearly shown in the 'Met?' column.

Y	Success criteria has been fully met
Y/N	Success criteria has been partially met
N	Success criteria has not been met

I have also referenced the stages of development in which each success criteria has been met/not met.

Success Criteria & Description	This colour indicates that the success criteria was removed during design (section <a href="#">2.5.1</a> ). Therefore, these criteria do not have any tests or development stages, and have not been met.
--------------------------------	---

It may be slightly confusing that tests and success criteria are referenced in a very similar way. I have tried to make it very clear when I am talking about a test to clear any uncertainty.

SC	Description	Met?	Development stages	Test evidence
G1	The interface must be simple and intuitive to navigate.	Y	All stages	Usability testing and stakeholder feedback
L1	The product has a login page	Y	2	Test <a href="#">L1</a> , <a href="#">L2</a> , <a href="#">L3</a> , <a href="#">L6</a> , <a href="#">R6</a> , <a href="#">R7</a>
L2	A user can only access data they are allowed to access	Y	2	Test <a href="#">L7</a> , <a href="#">L8</a> Ms Pearcey feedback
L3	Users can register themselves	Y	3	Test <a href="#">L4</a> , <a href="#">L5</a>
L4	The registration page validates information before entering into the database	Y	3	Test <a href="#">R1</a> , <a href="#">R2</a> , <a href="#">R3</a> , <a href="#">R4</a> , <a href="#">R5</a> , <a href="#">R19</a>
L5	Exam organisers will be able to view, create,	N	N/A	N/A

	and delete accounts, with password set upon first login			
L6	Email is sent to user upon account creation	N	N/A	N/A
L7	When logged in as a student, accompanist or music teacher, the time slot(s) they are involved in are displayed.	Y	12	Test P2, P5, P6, P7, P8, P12, P14 Ms Pearcey feedback
L8	When logged in as a student, accompanist or music teacher, and nothing has been scheduled, a message is displayed to say that no exams have been scheduled yet.	Y	12	Test P3
L9	When logged in as a student, accompanist or music teacher, there is a short message with the organiser's email included, to email them if they have any concerns.	Y	12	Test P9
L10	For students, if nothing is scheduled yet, another message is displayed : "If you have recently taken an exam, results will be displayed on the music notice board, or you will be contacted by your music teacher."	Y	12	Test P4
E1	Ability to load an existing schedule.	Y	10	Test P1
E2	Existing schedules are deleted at 22:00 on the (final) day of exam.	Y	10	Test I1
E3	There is the option to re-edit and re-publish a schedule, with the extra feature of only emailing those affected.	N	N/A	N/A
I1	For a new schedule, option to enter basic details such as start date.	Y	11	Test I2, R8
I2	Able to import and process a csv file of specified student information	N	(11)	Test I4
I3	Validation to check if student information is formatted correctly before data is saved to the server.	N	(11)	Test R11, R12, R13
I4	Accompanist availability can be added manually by organiser	N	(11)	Test I3, R9, R10
I5	Accompanists can be selected from a list and availability is requested	N	N/A	N/A
I6	Requesting availability sends accompanists an email	N	N/A	N/A
I7	Status can be seen while waiting for availability to be entered	N	N/A	N/A

I8	Acccompanists can log on and submit availability as 30-minute slots throughout the day.	N	N/A	N/A
S1	Creates a schedule that adheres to the TCL exam board regulations	Y	4, 5, 6, 9	Test S1, S2, R14, R15 Ms Pearcey feedback
S2	Schedule ensures minimal time is wasted.	Y	4, 8, 9	Test S3, S4 Ms Pearcey feedback
S3	As many exams are scheduled as possible	Y/N	4, 7, 8, 9	Test R16
S4	Schedule groups together each accompanist's exams as much as possible	Y/N	8, 9	Test S6 Ms Pearcey feedback
S5	Within each accompanist block, the schedule sorts exams by instrument family, then instrument, then grade.	Y	8, 9	Test S7, S8, S9 Ms Pearcey feedback
S6	Schedule adheres to accompanist availability	Y	7, 9	Test S5
S7	Schedule can determine if an accompanist has submitted enough availability	Y	8, 9	Test R17, R18
S8	Schedule is stored securely in server	Y	9	Test S10
S9	Opportunity to edit schedule before saving	N	N/A	N/A
S10	When editing, clashes are detected automatically	N	N/A (7)	N/A
S11	When editing, breaks are added automatically	N	N/A (6)	N/A
P1	The publish function sends mass emails to students, parents, accompanists, and music teachers.	N	N/A	N/A
P2	Emails send specific information with the option for the organiser to add some text.	N	N/A	N/A
P3	At 5pm the day before the exam start date, a reminder email is sent to students, parents, accompanists, and music teachers	N	N/A	N/A
P4	The publish function automatically creates a PDF of the schedule	N	(12)	Test P11
P5	There is an option to download and print the PDF	N	(12)	Test P10

This table shows all 38 success criteria:

- 18 have been fully met
- 2 have been partially met
- 18 have not been met (13 of which were removed in design, section 2.5.1)

The post-development testing has allowed me to provide evidence for meeting or not meeting these criteria. This table provides comprehensive information to evaluate the success of every criteria, but I have expanded on a few of these criteria below, providing further evidence or insights.

**G1** - This criterion is quite broad, but I believe that I have done more than enough to show that the interface is simple and intuitive. A thorough design of usability features in [2.3](#) and constant feedback from users during iterative development have contributed to this. The next section ([5.2](#)) will discuss this in more detail.

**L2** - The interview with Ms Pearcey provides further evidence for the success of this criterion:

- *Are you happy that only you can access the scheduling functionality and other admin features?*
- Yes, I am. It is good for security as it prevents anyone else from messing with the schedule.

**L7** - I believe that I have exceeded the standard set for this criterion, as the software has become more advanced than just displaying the time slots. It displays all relevant information, including an additional 'Teacher Name' column for clarity. Additionally, the schedule shown to the admin is split by breaks. These improvements make the system easier to use, and improve the functionality of this feature. During my interview with Ms Pearcey, she said that this feature is 'perfect' when it comes to displaying accurate and relevant information to each user.

**E1** - It would be more appropriate to name this success criteria 'able to view an existing schedule'.

**E2** - "Existing schedules are deleted at 22:00 on the (final) day of exam". My software does **not** do this, however I still believe that the criterion has still been met. After running into issues and re-evaluating this feature, I thought it would be more appropriate to include a 'Start New Schedule' button. This gives users more control without losing out on any functionality. In section [1.7](#), the justification given for this success criteria is:

*Once completed, existing schedules serve no purpose. They should be deleted to make space for the next schedule.*

My alternative, the 'Start New Schedule' button, is achieving the exact same goal as given above. It is a different solution, but in no way is it a worse solution. Therefore, I believe E2 has been fully met.

**I2, I3, I4** - These criteria have not been met, and were not developed due to time constraints. Therefore, accompanist and examinee data are manually entered into the database. This introduces the possibility of errors, and makes the system less robust, as it trusts that the user will enter valid data. To combat this, I introduced some validation. The software will check if the Examinees table is empty, and if so, returns an error. Test R20 shows that this check works successfully. This presence check makes the system slightly more robust, but unfortunately there is no other validation for this section. Removing these features has considerably reduced the robustness of the system.

**S1** - This is the only criterion where the test evidence is not unanimous. Tests S1, R14, and R15 are all completely successful, but test S2 is only partially successful. S2 was testing if breaks are scheduled within 10 minutes of the ideal break time. Each schedule has 3 breaks. However, it should be noted that test S2 was carried out 3 times: using the Spring 2023, Summer 2023 and Winter 2023 datasets. Therefore, there were 9 different breaks that were tested to check the success of S2 (3 datasets x 3 breaks). **Only 1 out of the 9 breaks failed the test.** This brings S2 very close to being fully successful. As all other tests (S1, R14, R15) were fully successful, I am happy to say that success criteria S1 has completely been met. To confirm this, I asked Ms Pearcey:

- *Do the schedules adhere to the TCL exam board regulations?*
- Yes, definitely.

**S2** - Ms Pearcey's interview provides more evidence for the success of this criterion.

- *Do you think it could have saved any more time or been more efficient?*
- I don't think so, no.

**S3** - This criterion is only partially successful, although I would say it is more successful than not. In most cases, it schedules as many exams as possible. This is done through prioritising exams correctly. The success of S2 shows that the schedule is very efficient and wastes minimal time, so naturally it schedules as many exams as possible. However, robustness test R16 shows that if accompanist availability is insufficient, the program stops scheduling exams, and does not schedule as many as possible. This issue is therefore why the criterion is partially successful.

**S4** - This is also partially successful, as my testing has found that the program groups accompanists very well, but usually has 1 exam out of place. I realised during my development that I had to balance 2 factors: scheduling accompanists efficiently, and grouping accompanist exams. This led to the introduction of the leniency threshold, which improved the grouping of accompanists significantly in stage 9 of my development. However, it has not completely solved this issue. Test S6 shows that exams are grouped perfectly by accompanist for only 1 out of the 3 datasets. The other 2 datasets lead to schedules with one exam out of place. Feedback from Ms Pearcey during our interview reinforces this idea:

- *Would the accompanists be happy with the schedules produced?*
- Essentially, yes. They occasionally have to hang around for a bit longer, but that is sometimes a good thing, it gives them a break. Most accompanists are grouped very well. The odd exam is out of place.

Section [4.1](#) provides more detail about her feedback for each dataset, but the above answer summarises it quite well. As most exams are grouped correctly by accompanist, but not all, this criterion has been partially met.

**S5** - Ms Pearcey has also agreed that exams are sorted correctly:

- *Do you think the exams could have been grouped any better?*

- [Taken straight from the interview transcript] Yeah. Yeah. That's great. I mean, it's almost more than it needs to be potentially, which is not a bad thing. Yeah. Most examiners also would like you for that.

She also raised an interesting point, that examiners would be very happy with the schedules produced here, as they find it easier when exams are grouped and sorted as I have done.

**S9, S10, S11** - These criteria represent the 'edit' feature, which was removed from the project during section [2.5.1](#). Stage 6 of my development created the 'addBreak' function, which will be a useful function to allow S11 to be met in future iterations of this solution. Stage 7 created the 'accompAvailable' function. This function can also be used to detect clashes when editing the schedule, which will contribute to S10 being met. It will form a modular part of the 'edit' feature in future versions of the system. These criteria have not been met, but the modularity of my solution means that it will be easier for these criteria to be met in the future.

## 5.2 Meeting the usability features

Section [2.3](#) outlines all usability features that should be developed. During all stages of development, I tried to include as many of the features as possible. By carrying out my own testing, and discussing in depth with all of my stakeholders, I have carefully tested each usability feature in section [4.4](#). The usability features also aim to meet success criteria G1: The interface must be simple and intuitive to navigate.

### 5.2.1 Consistency

A fundamental way of achieving a simple and intuitive interface is through consistency.

There are some specific techniques I used to achieve this:

- Colour coordination
- Hyperlinks
- Navigation bar
- Styling

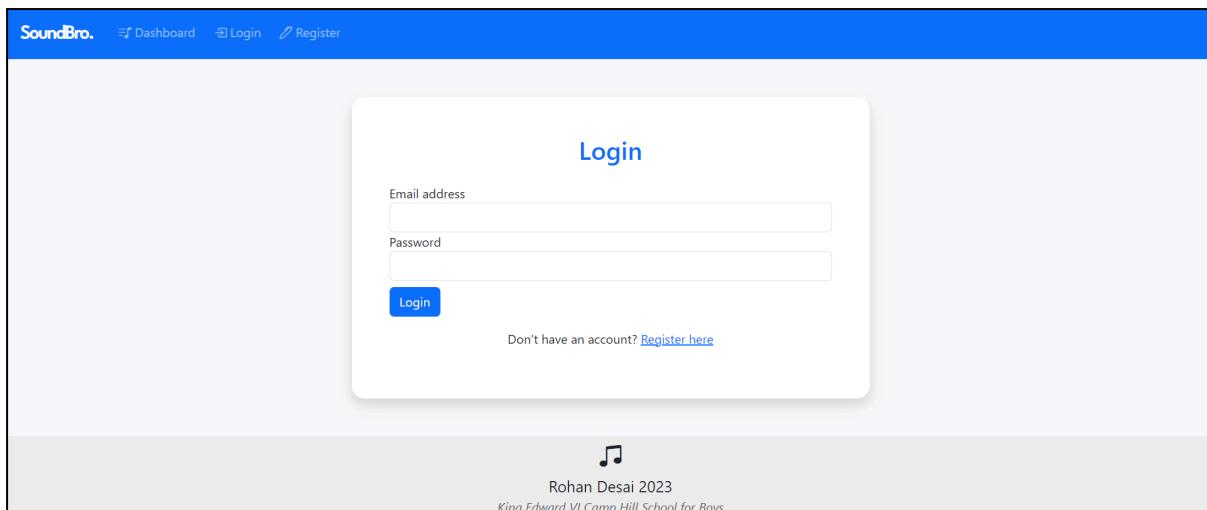
Colour coordination has been achieved by only using colours provided by the Bootstrap 5 framework. By limiting myself to only these colours, the interface looks simple and elegant.

The screenshot shows a web application interface with a blue header bar containing the logo 'SoundBro.', a 'Dashboard' link, and a 'Logout' link. The main content area has a light grey background. It displays two tables under the heading 'Exam Schedule | 26th Jan 2024'. The first table is titled 'Teacher Info' and has columns for Student Name, Instrument, Grade, Teacher Name, Accompanist Name, and Start Time. The second table is titled 'Accompanist Info' and has columns for Student Name, Instrument, Grade, Accompanist Name, and Start Time. Both tables show data for four students: Zaid Rassam, Leya Rahman, Xander Davis, and Lily-Marie Le Blanc, with Simon Palmer as their accompanist and start times ranging from 09:26 to 11:06.

Student Name	Instrument	Grade	Teacher Name	Accompanist Name	Start Time
You aren't teaching any upcoming examinees.					

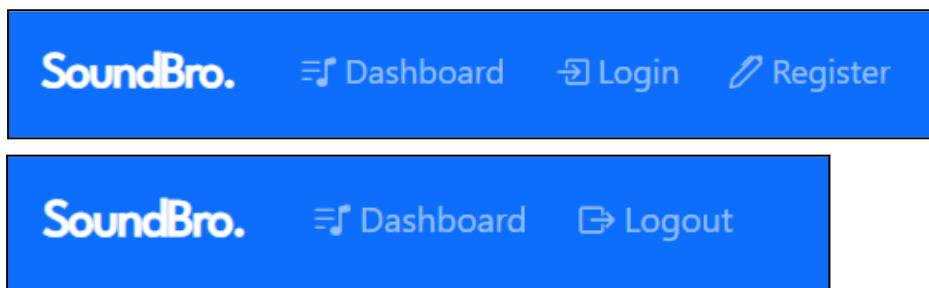
Student Name	Instrument	Grade	Accompanist Name	Start Time
Zaid Rassam	Trumpet	1	Simon Palmer	09:26
Leya Rahman	Trumpet	5	Simon Palmer	10:10
Xander Davis	Trumpet	6	Simon Palmer	10:28
Lily-Marie Le Blanc	Viola	3	Simon Palmer	11:06

There are only 2 colours used on this page (excluding shades of grey). Including more colours would be distracting and unnecessary.



This page only uses 1 colour, the same shade of blue seen on all pages. The login page provides another link to the registration page, which can direct any confused users. These links can be seen throughout the system. These pages also show consistency between the pages. By including the code for the navigation bar in a separate file to improve modularity, I was easily able to include the exact same navigation bar on every page. This makes users feel more comfortable with the interface, as a new page seems familiar to the previous one. It also makes it easier to carry out some actions, such as logging in/out or accessing the dashboard. The 'SoundBro' text is also a link to the index page.

The navigation bar changes when the user is logged in.



When logged in, the navigation bar displays links to the dashboard, and a logout button. There is no need to display a login or register button, so they have been removed. This reduces the choices that a user can make, simplifying the interface to only display what is relevant to the user.

However, I found that the navigation bar was not as useful as it could have been. Most users will be able to navigate to any page using the navigation bar. However, the admin also needs to access the admin dashboard quite regularly. Currently, the admin would click 'dashboard', and then click another link to access the admin dashboard. I could have made use of the navigation bar, putting a link to the admin dashboard here would make it accessible from any page. This change would be quite simple, as the session variable 'user\_type' will identify if a user is the admin or not, and using selection the link would only be displayed if the admin has logged in. Therefore, I think that the navigation bar is partially successful as an effective usability feature.

Another way of achieving consistency in the interface is by styling pages in a similar way. Below are screenshots of three different pages.

Welcome, Lorne!

This is the admin page. Click [here](#) to go to the music teacher / accompanist login.

**Schedule Music Exams**

[View Existing Schedule](#)

[Start New Schedule](#)

**Warning:** Starting a new schedule will delete the existing one.

**Schedule Music Exams**

Enter the start date

dd/mm/yyyy

*Please enter examinee and accompanist data directly into the database!*

[Schedule](#)

**Login**

Email address

Password

[Login](#)

Don't have an account? [Register here](#)

These pages all use the same container to show the relevant information. This container uses rounded corners and shadows to achieve a modern and professional look, and using this across many pages helps to achieve consistency across the system.

Furthermore, the name ‘SoundBro’ and its logo make the system more familiar to the user, allowing them to feel more comfortable with using it. Including the name on every page (in the navigation bar) enables this feature to be consistent across the system.

I believe that consistency across pages makes the system much easier to use. This usability feature is successful.

### 5.2.2 Error messages

Error messages are displayed constantly throughout the entire system. These messages are specific to the problem, which helps users to fix it. They are also emphasised in a red box, using a consistent style throughout the system.

#### Register a Student

Both names must be 2-50 characters, with no numbers or special characters (except hyphens)

#### Schedule Music Exams

The date must be in the future.

ERROR:

Elliot Drew has too many exams / is not available enough. They need at least 4 minutes more.

The above screenshots show just a selection of the possible error messages that can be displayed. I have carefully considered what the message should contain in each case for the information provided to be direct and specific to the error that caused it. The message for insufficient availability shows this quite well, as the accompanist’s name is given in the error, along with a rough estimate of how much more availability is needed.

Unfortunately, this estimate of the time is not very accurate. If I had more time, I would have liked to explore how to make this estimate better. Currently, it simply uses the number of minutes available, and number of minutes of exams for the accompanist. A better estimate would involve considering the specific availability of each accompanist, and checking if an accompanist's availability overlaps with another accompanist. A better result would provide Ms Pearcey with a more accurate message that allows her to make a well-informed decision. It would therefore simplify the communication between Ms Pearcey and the accompanist.

When some examinees have not been scheduled, their names are displayed in a message. Including their names allows the user to determine exactly who has not been scheduled and act accordingly.

The following examinees have not been scheduled: Leya Rahman , Xander Davis , Lily-Marie Le Blanc , Milaura Fernando , Maryam Rahman , Aliza Rayhan , Lauren Pumphrey , Jiayi (Eva) Chen , Henna Naveed , Doyinsola Oliyide , Akshay Suglani

During my interview with Ms Pearcey, I asked about the error messages:

- *What do you think about the error returned when accompanists have insufficient availability?*
- That's, again, very good. There would potentially be one thing that could make it a bit more useful. When it says who hasn't been scheduled, it would be good to know how much total exam time is needed to schedule everyone. That would help me when thinking about booking another day or half-day, because I would know how much is left to schedule.

Ms Pearcey said that it would be helpful to know the total duration of the unscheduled exams. This would be quite simple to calculate, it would require a change to the function recalculateOrder. The array 'accompIDs' already stores the number of minutes of remaining exams for each accompanist. I would simply have to add up this value for each accompanist in accompIDs, and display this number. In the future, this addition would allow messages to be more useful to Ms Pearcey.

The error messages are quite relevant to the issue causing it, but some messages could be more informative. Therefore, this feature is partially successful.

### 5.2.3 Helpful messages

In addition to error messages, I have included several important messages to clarify information, direct users, or prevent errors. Some examples are shown below.

The schedule has been published!

All examinees have been scheduled!

Colours are used in these messages to convey the purpose of the message. Green indicates success, blue is for emphasis, and red is for errors (seen in the previous section). These messages clarify the result of an action, for example clicking ‘Publish’ leads to the first message.

## Welcome, Lorne!

This is the admin page. Click [here](#) to go to the music teacher / accompanist login.

This message directs the user to the correct location, making it easy to navigate to the desired location.

Email address

Parent email address

We'll only send you and your parent an email when the schedule is created.

Password

Password must be at least 8 characters long and include at least one uppercase letter, one lowercase letter and one number.

**Warning:** Starting a new schedule will delete the existing one.

These messages help to prevent unwanted issues. The first image is taken from the registration page for students. The message about emails informs the user of the purpose of collecting emails, and assures that they will not be bombarded with spam emails. This message was included as a result of stakeholder feedback from my system testing in stage 3 (section [3.3.3](#)). The message about the password explains how secure the password should be, which improves the chances of users providing correct details the first time. The warning message prevents the user from accidentally emptying the database. These messages are succinct, simple, and informative. By informing the user, they feel more comfortable with using the system, and are more likely to trust it. This results in a positive experience for the user. I contacted my stakeholders to ask about the success of this feature.

*Question: Is the on-screen help useful?*

**Ms Pearcey:**

Yes, they are quite helpful. I don't think I would want any more messages.

**Cyrus:**

It is very useful, any more text and it might get a bit overpowering. You could consider having that little question mark icon with an explanation inside.

**Darius:**

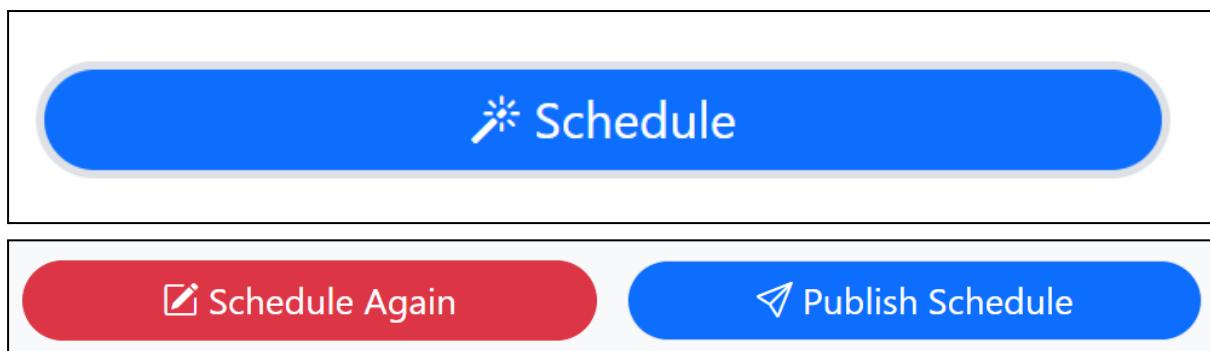
Yes. It's very direct and honest.

All of my stakeholders said that these messages were useful and clear. Therefore, the feature of on-screen help is fully successful as an effective usability feature.

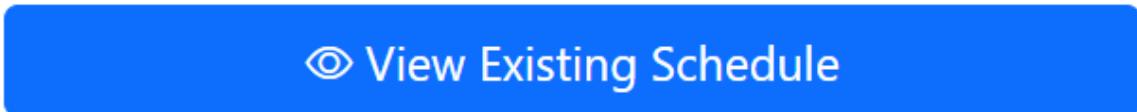
In the future, I could try to implement Cyrus' suggestion of a question mark icon. This would display text if the cursor hovers over it, which prevents all text from being displayed on the screen. This would simplify the interface, but currently I do not believe it is necessary. It is likely that further improvements to the functionality will lead to more of these messages. Only if more messages are included then I would consider using this technique, in order to prevent overwhelming the user.

#### 5.2.4 Buttons and icons

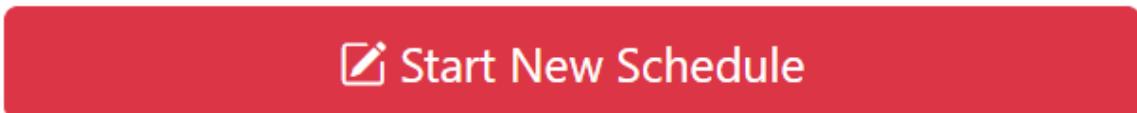
The system has several important buttons, which have been emphasised to improve usability. The most important buttons are 'Schedule' and 'Publish Schedule', as these execute quite significant functions in my program. Therefore, I have taken the necessary steps to ensure that these buttons are clear and easy to press.



By highlighting the buttons in the primary colour of the website (blue), it stands out on the page. The button itself is quite large compared to other text, and its rounded shape gives a professional feel. The 'Schedule Again' button carries out the same function as 'Start New Schedule'. Therefore, it is coloured in red for consistency. The red colour also conveys the danger of pressing this button (it empties the BasicInfo and Schedule table).



⌚ View Existing Schedule



📝 Start New Schedule

All of these buttons use icons. Using Bootstrap, I was able to introduce icons in these buttons as well as in the navigation bar. This allows users to easily identify the purpose of a button, as they can see the icon and recognise what it does. The magic wand in the schedule button implies the simplicity of the system. I asked Ms Pearcey the following questions to evaluate this feature's success:

- *Is the 'schedule' button easy to press?*
- It's very easy and clear. Easy to press.
  
- *Is the 'publish' button easy to press?*
- Yes. It's very clear.

Ms Pearcey seemed very happy with this button and how it has been displayed. I am therefore confident that this feature is fully successful.

### 5.2.5 Mobile compatibility

As students will be using this system to view their exam times, they are highly likely to want to log on with their mobile phone. Therefore, I used the Bootstrap framework, which makes it easier to make mobile-compatible websites. By using Bootstrap classes as much as possible, I tried to make this website very easy to use on mobile devices. However, my post-development testing (section 4.4) showed that this feature is only partially successful.

The image displays two side-by-side screenshots of the SoundBro website. The left screenshot shows the login page with a blue header containing 'SoundBro.', a 'Dashboard' link, and 'Login' and 'Register' buttons. Below the header is a form with 'Email address' and 'Password' fields, a 'Login' button, and a 'Don't have an account? [Register here](#)' link. At the bottom is a footer with a musical note icon, 'Rohan Desai 2023', and 'King Edward VI Camp Hill School for Boys'. The right screenshot shows the dashboard with a blue header containing 'SoundBro.', 'Dashboard', and 'Logout' links. The main content area has a title 'Dashboard' and a welcome message 'Welcome, Aakshat!'. It includes a note about scheduling and contact information for Ms. Pearcey. At the bottom is a footer with a musical note icon, 'Rohan Desai 2023', and 'King Edward VI Camp Hill School for Boys'.

The website looks slightly less polished when simulated on a mobile device. The navigation bar becomes quite large, taking up a considerable amount of the screen space. The website is still fully functional, and the content of most pages conforms to the screen size appropriately.

**SoundBro.**

Dashboard

Logout

## Dashboard

Welcome, Aakshat!

Exam Schedule | 26th Jan 2024

Student Name	Instrument	Grade	Accompanist Name	Start Time
Aakshat Kumar	Classical Guitar	4		15

Email Ms. Pearcey at [admin@email.com](mailto:admin@email.com) if you have any concerns.

Rohan Desai 2023  
King Edward VI Camp Hill School for Boys

**SoundBro.**

ashboard

ogout

## Dashboard

Welcome, Aakshat!

Exam Schedule | 26th Jan 2024

Student Name	Instrument	Grade	Accompanist Name	Start Time
Aakshat Kumar	Classical Guitar	4		15:48

Email Ms. Pearcey at [admin@email.com](mailto:admin@email.com) if you have any concerns.

Rohan Desai 2023  
King Edward VI Camp Hill School for Boys

Unfortunately, the most important piece of data, the start time, is not directly accessible when logging in. Scrolling horizontally reveals the time, which may not be directly obvious for the user. It also reveals that the rest of the website abruptly ‘stops’, making the interface look quite rough and unrefined.

Unfortunately, my iterative testing did not test my system on a range of screen sizes. Therefore, I was unaware of these issues until my development was complete. If I was to do this process again, I would make sure to test on many different screen sizes throughout the development process. This would help me to detect and remove any errors or inconsistencies with the interface.

I believe this feature is partially successful, as the functionality still works well on the phone, and most pages are reasonably easy to use. However, in some areas the interface looks unpolished and not as intuitive as I had designed it to be.

### 5.2.6 Further changes

I asked my stakeholders if there was anything else they would want to be implemented to make the system more usable.

#### **Ms Pearcey:**

The only thing that might be nice would be to save the previous schedule somewhere so that if you did accidentally delete it, you could see it or you could compare schedules potentially.

#### **Cyrus:**

I think it's quite usable already, the icons are a nice touch. I'm not sure how difficult this would be, but maybe you could include some accessibility features to make it usable for everyone.

**Darius:**

The footer you have on every page should be at the bottom of the screen, not halfway up. It would make it look a lot more professional. I was thinking that you could use that 'sign in with google' thing instead of making logins, because that way you wont forget this password. I don't think a year 7 would remember a password that he uses once, maybe twice a year. That's just an idea though, overall it is very easy to use.

All of these features mentioned would definitely improve the usability. Ms Pearcey suggested saving previous schedules to the server, which would be a good idea. This would allow her to reuse previous data, or compare different schedules, which would improve the efficiency of the scheduling process. This is a feature that I was considering earlier in my project, and I was hoping to implement this in the future anyway. With this in mind, I designed my database in such a way that BasicInfo stores some data about the entire exam cycle, so storing multiple records in this table could be a reference to each saved schedule. This would be a starting point, but my database design is flexible enough for this feature to be possible in the future.

Cyrus suggested including accessibility features to make the system usable for everyone. This is an important point, as my system does not have much support for those with visual impairments. I could adjust my program to be more compatible with screen readers. This includes:

- Using alt text on images
- Using more appropriate HTML tags
- Using ARIA roles in HTML

These techniques will provide more information to the screen readers, enabling any text-to-speech to be more accurate. For example, using `<div>` repeatedly is less descriptive than using `<p>` or `<h2>`. When a screen reader sees the `<table>` tag, it knows to let the user navigate horizontally and vertically through the table. I have used this when displaying all schedules, which makes the system more accessible to everyone. I would also like to use more contrasting colours to make the text easier to read for people with colour blindness.

Darius suggested some minor changes to the interface, but made an interesting point about logging in to the system. He said that most students are unlikely to remember a password that they will only use once or twice a year. I completely agree with this statement, and agree with his suggestion to implement 'sign in with google', otherwise known as Google OAuth. This would make my system easier to use, as users will be able to log in using their google account, and not have to make a separate account for this website. It could also simplify some other future developments to the system, such as email notifications. I have discussed this in the next section.

## 5.3 Further development

18 success criteria have not been met, and 2 have been partially met. This section will discuss how future development could address these criteria.

### 5.3.1 Gathering inputs

I2	Able to import and process a csv file of specified student information	N
I3	Validation to check if student information is formatted correctly before data is saved to the server.	N
I4	Accompanist availability can be added manually by organiser	N
I5	Accompanists can be selected from a list and availability is requested	N
I6	Requesting availability sends accompanists an email	N
I7	Status can be seen while waiting for availability to be entered	N
I8	Accompanists can log on and submit availability as 30-minute slots throughout the day.	N

These criteria involve inputting accompanist and examinee information into the system. Unfortunately, the criteria were removed during this project, due to the complexity of their execution compared to the time available. During further development, I2-4 would be the first criteria I would try to meet. Meeting these criteria would make the system much more usable and functional, as Ms Pearcey would be able to enter the relevant data in a much simpler way than the current system. Currently, examinee and accompanist information is entered directly into the database. This has no validation, requires a detailed knowledge of the database structure, and involves converting each user into the relevant StudentID / SupervisorID.

By developing a feature on the website to enter and process this data, all of these issues could be solved. Examinee information would be submitted as a CSV file, as specified in section [2.4.1](#). The accompanist information would be entered directly into a form in the website. During section [2.5.4](#), I designed algorithms for these features in detail, developing these algorithms successfully would result in I2-4 being met. Each aspect of the CSV file would be validated, including checking each element to see if it exists in the database as a valid foreign key (eg. checking if a student exists in the Students table). This would make the system more robust, and specific error messages would be helpful and simplify the process even more. The file would require Ms Pearcey to enter a person's email address, which will automatically be linked to their StudentID / SupervisorID. This reduces the workload for Ms Pearcey, as entering emails are much simpler than entering the correct IDs.

Including a page for accompanist information would also mitigate the difficulty of handling IDs. A drop down list would only display the names of registered supervisors, which automatically links to the SupervisorID, abstracting all complexity from the system. These

changes would significantly improve the system's functionality and reduce Ms Pearcey's workload.

Developing I5-8 is less important, but would still improve the website's functionality. Instead of making Ms Pearcey submit accompanist availability, they would log on and do it themselves. This reduces her workload, and takes advantage of the login system. This would directly populate the database.

Ms Pearcey would view a list of registered supervisors, and select all accompanists she needs for the current exam cycle. Clicking 'submit' would send an email to everyone in the list and update their dashboards with the relevant form. After logging in, they would be able to submit availability as 30 minute blocks for the specified day. During this time, Ms Pearcey should be able to view who has and has not entered availability, and when she is happy she can click 'next'. This button removes the form from each accompanist's dashboard, and moves on to entering examinee information. This simplifies the process of submitting availability, making it easier and quicker to move on to the next stage. This process can therefore lead to faster scheduling.

### 5.3.2 Making a PDF

P4	The publish function automatically creates a PDF of the schedule	N
P5	There is an option to download and print the PDF	N

The next most important feature to include would be to create a PDF of the schedule, shown as success criteria P4 and P5. I designed this feature, and attempted to develop it. Unfortunately, complications with the DOMPDF library were out of the scope of the project and I could not continue with its development. In the future I would try using a more common library, such as TCPDF or FPDF. There is more online support available for these libraries, and extensive documentation to help me develop this feature correctly. These libraries are much more stable than DOMPDF, so any errors with the libraries have been fixed, and any errors I find are likely to have already been solved on Stack Overflow! As I have successfully produced a table of the schedule already, this process would involve using the same table in the file. A button would be displayed clearly on the screen, directing users to download the PDF of the schedule. Ms Pearcey would be able to print this out and display it on the music notice board, for everyone to see easily. A survey conducted during my analysis ([1.3.2](#)) showed that 95% of examinees find out about their exams by seeing the timetable on this notice board. Including this feature will automate the task of producing the file, reducing Ms Pearcey's workload. It will also allow my system to integrate into the school easily, by supporting the method which 95% of examinees are familiar with. Ms Pearcey herself said that this feature was quite important to her, but unfortunately I did not effectively prioritise my tasks and could not implement this in the time available. Therefore, further development would focus on building this feature with a much higher level of urgency.

### 5.3.3 Optimising the scheduling process

S3	As many exams are scheduled as possible	Y/N
----	---	-----

S4	Schedule groups together each accompanist's exams as much as possible	Y/N
----	---	-----

S3 is not fully successful because robustness test R16 shows that if accompanist availability is insufficient, the program stops scheduling exams, and does not schedule as many as possible. In order to solve this problem, I would adjust the scheduling algorithm. Currently, the process of adding an error message to the variable \$error is causing the scheduling to stop. Instead, I would add the error message into a different variable to prevent the loop from being broken. The algorithm will therefore iterate through the entire day, scheduling exams whenever possible to maximise the number of exams. Pre-existing code will ensure that the schedule is still valid and accompanist availability is still being met. At the end of the function, this additional variable will be returned, so that the error message for accompanist availability is still displayed. This allows S3 to be met, as the schedule will be able to schedule as many exams as possible in any situation.

Solving S4 is a bit more difficult. I already encountered issues with accompanist grouping in my stage 9 system test ([3.9.2](#)). A fundamental part of the solution, recalculateOrder, was being too strict with the way it sorts accompanists. This led to schedules being produced with an alternating pattern of accompanists. Therefore, I introduced a 'leniency threshold' to find balance between maximising accompanist efficiency, and grouping accompanists together. If an accompanist's free time is above the threshold, they have enough free time, so they stay in their original order. If below the threshold, they do not have much free time, so their order changes to sort accompanists by priority. This solved my problem, but was not a perfect solution to the problem. My post-development testing found roughly 1 exam per schedule in an incorrect order, as this exam was not in an accompanist group. This shows that my system needs some refining.

Currently, I use a threshold. A single value that is used to decide to sort or not to sort. However, I could adjust this system to be less binary, and more of a gradient. My algorithm could use the free time in some mathematical function, so that a larger free time gives a number closer to zero, and a lower free time gives a larger number. This 'number' would be used in the PHP usort() function. To provide some more context, usort() allows me to create my own function to determine the order of 2 elements. This function returns a positive number if the first element should go after the second, and vice versa for negative numbers. It returns 0 if the order should remain the same. The free time of each element can be input into some mathematical function, such as  $y=k/x$  (where k is some constant). The output of this function can be multiplied by the difference of the 2 free times. This method would try to sort exams by accompanist free time, but the larger the free time, the lower the chance of the order changing. This would need some testing to find the correct mathematical function to use, but it would be much more sensitive to the free time than a single threshold value. This could potentially improve the algorithm used for grouping accompanists, so the schedule will be more likely to meet Ms Pearcey's requirements and save accompanist time.

It is possible that the optimal value of the threshold will vary depending on the inputs provided, so each schedule may have a different one. Another option would be to simply give control to the user, and display a slider for the user to choose their 'leniency' of accompanist grouping. This slider would determine the value of the leniency threshold, and

allow the user to adjust the inputs as they please. I briefly mentioned this idea to Ms Pearcey, who said that it would be overkill, as the user would have to input too much information into the system. Therefore, my further development would try to fully meet S4 using the first method.

### 5.3.4 Editing manually

S9	Opportunity to edit schedule before saving	N
S10	When editing, clashes are detected automatically	N
S11	When editing, breaks are added automatically	N

My post-development testing and interview with Ms Pearcey revealed that this function would be quite useful. The feature to manually edit the schedule before publishing it allows Ms Pearcey to adjust it to her liking. Any small errors in the schedule can be fixed easily, and if any changes need to be made, this is also incredibly simple. It makes the system easier to use, as it gives control over the entire schedule to Ms Pearcey. If she finds any issues with the scheduling algorithm, she has the power to fix these mistakes herself. During our interview, she said this:

***"Would you change anything about the scheduling functionality?"***

*I don't think so. Maybe in the future, if you were developing this commercially, you could give the option to manually swap 2 students. After making the schedule, I might get to the stage and then somebody will say to me, I can't do that time because of this and that, and sometimes get extra restrictions put in. If at that point you can take this nice display and just make manual changes, that would be ideal.*

Developing this feature would evidently improve the functionality of the system and reduce Ms Pearcey's workload. It enables the system to be much more flexible, the order of the schedule can be adjusted to fit the 'extra restrictions' that may be introduced. This would be developed by making the timetable into a drag-and-drop list. Each time a change is made, a function would run to recalculate the breaks and detect clashes (S10 and S11). The schedule **must** meet TCL regulations, so breaks will be added automatically to ensure the schedule is always valid. This will be done by iterating through each exam in the provided order, and adding breaks whenever the regulations are about to be broken, or whenever it is time for a break. This would be the same logic used to schedule breaks in my schedule function. If a break needs to be scheduled, the addBreak function will run. This function has already been created and tested (see development - stage 6) so the process of meeting S11 is simplified. The start time of each exam will be updated accordingly in the database.

The algorithm should assume that Ms Pearcey knows better when it comes to accompanist availability, so it will not make any more changes to the schedule based on availability. Instead, the function will iterate through each exam and check if their accompanist is available for the entirety of that exam. This check will occur by calling the function accompAvailable (see development - stage 7), which has been developed and tested already. If accompAvailable returns false (accompanist is unavailable), the interface will display a warning message, as a clash has been found. This alerts the user without taking

control of the schedule, so Ms Pearcey is still able to adjust the schedule however she likes. These functions will ensure that the schedule is always valid and will provide information about any issues with availability. The feature of editing schedules manually would be incredibly useful to Ms Pearcey, as any issues with the scheduling algorithm, or any last-minute changes can easily be fixed by dragging and dropping exams into the desired order.

### 5.3.5 Emailing

L6	Email is sent to user upon account creation	N
I6	Requesting availability sends accompanists an email	N
P1	The publish function sends mass emails to students, parents, accompanists, and music teachers.	N
P2	Emails send specific information with the option for the organiser to add some text.	N
P3	At 5pm the day before the exam start date, a reminder email is sent to students, parents, accompanists, and music teachers	N

The above success criteria can be met by incorporating email functionality into the system. This was originally planned, but removed from my design due to several security concerns and issues with complexity (see [2.5.1](#)). I came to the conclusion that the best method of sending emails would involve using Google OAuth, which lets users sign in to the system with their google account. This is very secure, as it retrieves data that is stored in Google's servers, and it also works with PHPMailer, the library for sending emails. This option could be integrated with the login system, so users can either log in normally or 'sign in with google'. If the admin was to log in, I can use PHP to authenticate the user with Google OAuth. Later in the system, I could send an email using PHPMailer with OAuth authentication. This can be done using an 'access token', which ensures that my PHP script is authorised to send emails on behalf of the user. This may be an email for account creation, requesting availability, or informing everyone of the schedule.

This would also improve the usability of my system, as it removes the need for users to remember their passwords and makes it much easier to log into the website. This was suggested as a possible improvement to the usability of the system by Darius, who said:

*I was thinking that you could use that 'sign in with google' thing instead of making logins, because that way you wont forget this password. I don't think a year 7 would remember a password that he uses once, maybe twice a year.*

Darius made a great point, most users will rarely log in so will most likely forget their password, making it harder to access the system. Google OAuth can solve this problem, as the vast majority of people will know their google account password easily. Therefore, using Google OAuth can be a secure way of sending emails to users, as well as making the website more usable.

All of the changes and improvements suggested in this section can be used in further development to meet all unmet / partially met success criteria. Some features were removed during my project, and were not covered by any success criteria. These features have been discussed in the following section.

## 5.4 Limitations

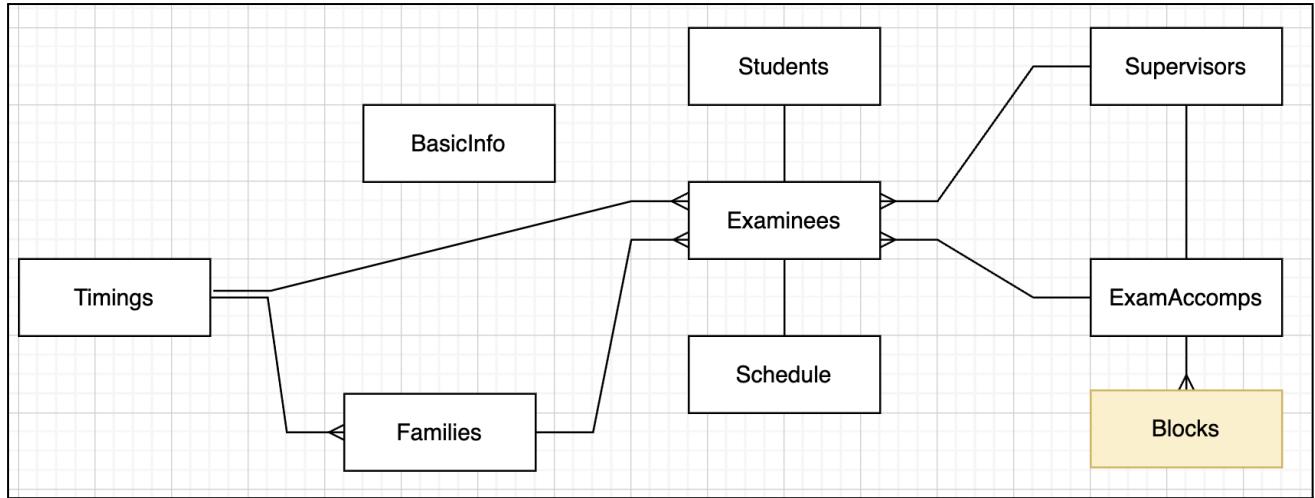
### 5.4.1 Schedules for 1 day only

One main limitation of my solution is that it only schedules exams which last one day. If too many examinees are inputted, it will schedule as many as possible and then stop scheduling by 5pm. I originally planned that it would not stop scheduling at the end of the day, instead it would reset the time to 9am for the next day, and carry on until everyone is scheduled.

Unfortunately, I faced some difficulty when trying to design the database for this. I was not sure how to store accompanist availability if the number of days is variable. Therefore, I removed this feature during section [2.4](#). This prevented the project from becoming too complex, but makes the solution less useful to my stakeholders.

To avoid this limitation, I could try to change my database appropriately. Currently, the times 9am to 5pm are stored in 16 30-minute blocks, so there are 16 fields for availability in ExamAccomps. Assuming that exams will never be longer than 3 days, I could simply include fields for day 2 and day 3, so each record would have 48 fields for availability. This would work in principle, but is quite computationally intensive and would take up lots of storage space. It is also rather inelegant, as this method clearly does not scale well. Each additional day requires 16 more fields, putting more strain on the system and reducing performance. In the future, if someone wanted to change the system to schedule exams over 4 days, they would face significant difficulties with managing storage space and improving performance. This method also stores lots of redundant data. If exams are scheduled for one day only, availability for all other days will still be stored, making the system quite computationally intensive. This could lead to the website crashing, or it could limit the types of devices that can schedule exams.

All of these problems show that this approach is not maintainable. Therefore, another approach should be taken to store availability over multiple days. I could change the database so that availability is recorded in ‘blocks’ (eg. 09:00 - 14:45). However, someone may have different availability on different days. Therefore, each accompanist should be able to store many blocks of availability. Therefore, a ‘Blocks’ table should be created to store this information. One record in this table would store the start and end time of the block, as well as the AccomplID of the accompanist. This ID would act as a foreign key to connect to the rest of the database. These ‘start and end times’ would also store the date.



The entity-relationship diagram above shows a possible change to the structure of the database. The new 'Blocks' table is shown in yellow. Using this method would also remove the limitation of availability only being stored in 30 minute blocks, as the start and end times can be at any time. Therefore, accompanist availability can be stored more accurately, which maximises the time each accompanist is available for and makes scheduling more efficient. This approach would involve no redundant data, use a much simpler structure that is normalised to 3rd normal form, and allow availability across many days to be stored accurately. This method scales very well, and prevents performance issues, making it very maintainable.

In order to schedule exams over many days, I would also change the schedule function, so that the time is incremented to 9am of the next day whenever eod = true. The StartTime field in 'Schedule' would store the date as well as the time, making the database fully compatible with these changes.

#### 5.4.2 Compatible with this school only

Another limitation is that the solution is only made to schedule exams for this school. This is because I have tailored the solution to try to meet Ms Pearcey's specific requirements. This includes: specific break times, a fixed start time, and the concept of a 'school' accompanist. This has been explained fully in [1.1.3](#). Originally, I was planning to make a system that schedules exams for schools across the country. However, I faced some security issues and underestimated the complexity involved in doing this. Each school would need some sort of login, and each user would only be for a single school. This would require lots of inputs into the system, and an exceedingly complicated database to store all of this data. Therefore, I removed this feature towards the end of my analysis. This allowed me to focus my efforts on scheduling exams for Ms Pearcey, specific to her requirements. As a result, some parts of my system are difficult to maintain if support for many schools were to be included in the future. For example, her email address is hardcoded into the program.

To make this solution compatible with many schools, the scheduling algorithm would have to take some more inputs. Firstly, the admin would set the ideal break times. This would determine what value should be used in the logic of the nextBreak function. Also, the start

time for each day would be entered into the system, to decide what time should be initialised to. These inputs would be validated appropriately. By introducing more inputs into the solution, the scheduling algorithm can work effectively for many different schools.

The database would also store some metadata for each school, such as the admin's email address. This email address is displayed at the bottom of the dashboard, and is used to give the admin exclusive access to scheduling functionality. I would change my solution by retrieving the email from the database, instead of hardcoding it into the program. This allows different emails for different schools to be used correctly, preventing errors related to managing multiple schools.

#### 5.4.3 Other limitations

During my development, I realised that it would be great if results were also displayed on this website. Ms Pearcey would be able to enter the results for each examinee into the website, and the dashboard would retrieve this data from the database, and display the result for each student. This would require changes to the database, as the Examinees table would include an additional field called 'Result'. This would store 'distinction', 'pass', 'merit', or 'fail'. I would likely use a similar method to publishing the schedule, so BasicInfo may have a field called 'PublishResults'. If Ms Pearcey clicks a button, this field will be set to true. This will cause the user dashboard to display the results. The information displayed would only be relevant to them only, so someone wouldn't be able to view another person's results to respect their privacy. The students would also receive an email to inform them. This would improve the efficiency of communicating results, as well as making it easy to discreetly inform everyone.

Another limitation with the system is that students are not asked for availability. Asking students would allow any clashes to be detected **before** any scheduling is carried out. Most students will be completely available, but some may have a school trip, dentist appointment or some other commitment. Therefore, I could include availability for students by implementing it in the same way as accompanist availability. All data would be validated appropriately to prevent errors occurring.

## 5.5 Maintenance

I believe that my solution is quite maintainable. The previous section has already identified some possible changes that could be made to improve maintainability, but on the whole I feel that it is quite simple to make changes to the system.

I have developed this code with a significant level of modularity. The scheduling functionality alone is made up of 5 functions, and each one is self-contained and has been designed, developed and tested in isolation. There is one additional function, the schedule function, which calls these functions to allow all modules to be integrated into the solution. These functions are:

- nextBreak
- meetingRegulations
- addBreak
- accompAvailable
- recalculateOrder
- schedule

For each function shown above there is a corresponding flowchart in [2.5.2](#). There is also a corresponding stage of development for each, with comprehensive unit testing to find and fix errors. After discussions with Ms Pearcey, I learned that regulations are not fixed, and can change regularly. Therefore, I have developed a system with lots of modularity so that edits can be made easily. If the TCL regulations were to change in any way, editing the meetingRegulations function would implement these changes consistently across the entire system. Additionally, if the ideal break times were to change then nextBreak can be modified easily. By using modularity extensively in my solution, changes to the solution can be made easily and effectively.

During development, whenever repeated code occurred it led to the development of a new function. Creating these functions improved the structure of the code and its modularity. Using subroutines instead of repeating code allows any edits to be made easily, as a change to the function will consistently update all parts of the code where it is called. The following functions have been created to further improve modularity and maintainability of the scheduling functionality:

- getAccompField
- getExamineeInfo
- timeDifference

All scheduling functionality was also designed and developed separately from the other large modules. For all other modules, I used this principle of modularity to make debugging easier and ensure consistency throughout the system. I noticed that some blocks of code were being repeated on every page, so I added them into a separate file and used 'include' to reference this one file. These files were stored in the 'includes' folder:

- db-connect.php
- footer.html
- navbar.php

```
<?php include 'includes/navbar.php' ?>
```

These 3 files were called by every page, so using an external file prevented repeated code, improved modularity and simplified any changes that had to be made. They also allowed the interface to achieve a consistent look, improving usability. Modularity has been implemented throughout the solution to improve maintainability, and I do not believe that there is any way to further improve the system's modularity.

Another way of making a maintainable system is through the use of comments. All of my code has been annotated appropriately to clearly describe what each part does. This makes it easier for other users to understand the code and edit it as necessary. My final code (see [Appendix](#)) as well as my development evidently shows that comments have been used throughout the entire solution.

During development, I have tried to name all variables and structures appropriately, and I believe that I have done this successfully. Arrays such as examineeIDs and accompIDs are just some examples of identifiers that make the purpose of the array clear. This can also be seen in my database, as each table and field name is direct and easy to understand. For example: Students, Supervisors, and Schedule are some tables in my database. Anyone who accesses my program will be able to gather a significant amount of information from the identifiers alone. This makes it easier for anyone to understand and make changes to my system, so the solution is more maintainable. There are a very small number of exceptions to this. The function instrumentSort has 2 parameters called a and b. This can be quite confusing for someone who is not familiar with the usort() function. Changing these names to 'elementA' and 'elementB' will make it clear that these are 2 elements from an array, allowing the reader to develop a better understanding of the system. Apart from this function's parameters, I believe that all identifiers have been appropriately used to make the solution maintainable.

My database is normalised to the third normal form (3NF), so data redundancy is minimised through the elimination of transitive dependencies. Each piece of data is stored in only one place. This reduces the chances of inconsistencies and makes it easier to update information uniformly across the database. For example, it is likely that the duration of some exams may change in the future. By adjusting the relevant record in the Timings table, the entire database and system will be updated to reflect this change. Poorly normalised databases may have inconsistent data, which leads to logic errors and makes maintaining the database exceedingly difficult. Databases in 3NF are also more flexible to changes, so it is easier to add a new field or table without disrupting the existing structure. A great example of this can be found in [5.4.1](#), where I suggested introducing another table to allow exams to be scheduled over multiple days. As my database is in third normal form it is much easier to make changes to the system. Through this efficient database structure, my solution is more maintainable.

# 6 Appendix

## 6.1 Full stakeholder responses (conducted during analysis)

### 6.1.1 Ms Pearcey interview transcript

■ Me  
■ Ms Pearcey

Can you describe the current system used for scheduling music? Great exams, and your role in that. Okay. So I get all of the entries in, which gives the names, grades, instruments. And I agree dates that we would like to the exam board and we request them, I have to then jump the gun and presume that we're going to get the dates that we want and find out or check, who needs an accompanist for their exam and then I've got to work out who's going to do that accompanying. So normally we'll try and use just one person if we can. So Mr. Palmer now and maybe myself sometimes to fill in some gaps and then I have to check his availability. Some of the music teachers will accompany their own students, so I need to check their availability for the day or days we've got. And then I have got a spreadsheet here which I'll let you take some pictures of if you want to. So I actually get the data into a Google form these days, but then I can put it through to another sheet and If I turn it on here. Yeah so then I've got V-lookups that will look up the grade and give me the length of the exam. Some of the exams have different lengths. So I have to do that manually because I'm only looking up, obviously on the one criterion not both of them, but I know for example that piano exams are different and singing exams are different.

Okay, so are the exam times based on instrument or grade? They are both, mainly on grades, some instruments have different timings for the same grade. So singers have something like 15 minutes for each grade. Grades three to five, whereas other instruments, might be 14 and 17.

Okay. Is it dependent on the exam board? Yes, that changes as well. So, separate spreadsheets for each board. In one day can you do different exam boards? Its separate sessions. So you'd have to run each one separately because they have nothing to do with each other but yes I can. So I could have an examiner here in any examiner there and then I'd have to juggle, say if Mr. Palmer was playing for both exams. Maybe he'd do the morning exams over in the girl's school for ABRSM and then come over here in the afternoon and do Trinity exams.

So the examiners that they exam board-specific. Yes. And how do they come here? So I book a session, I'll try and request the dates that I want. Hopefully at some point they confirm to me that those are the dates I'm gonna get. They've then got their own systems where I have to enter the timetable but it doesn't help me to timetable. It's just a process that I have to go through to put the timetable online for them, which I'll show you in a minute as well, but basically it's just drag and drop things around and it sort of helps a little bit for ABRSM because it puts in breaks for you. For the other one [trinity], I have to manually create the breaks. So when are the breaks normally? The boards have got their own guidance on what this should be, but roughly speaking it's a maximum of two hours in the first session.

And then you have a break of either 10 or 15 minutes depending on the exam board. And then you have about another 1.5 to 2 hours, taking you through to lunch. For Trinity the lunch is 60 minutes. For ABRSM it's 65 minutes and then you have another break mid afternoon.

**So normally, do you have different exam boards on separate days?** In theory you can, but it's just too complicated.

**Apart from availability, do you create the schedule based on certain factors, for example by grouping an accompanist's exams together?** Yes, I'd need to group all of one accompanist's exams together as much as I can. If it's one of the visiting teachers, then obviously all of their own pupils would be grouped together. For Mr. Palmer coming in, this session we've got coming up for example, I've got him with some exams before lunch or some straight after.

So they are broken by the lunch break. That's just really to accommodate everything else that was the best way of doing it. **So, the way you do it, it's based on availability, but it's also the accompanist's efficiency, just to make sure that they're not too spread out?**

Yes. You don't want them to be having to hang around because they're not being paid for that time. Equally though, if you've got a really long session, it's good to put in a bit of a break. Yeah, accompanying is quite hard work. So if there's instruments that don't need accompaniment like piano or guitar, I might sometimes stick a few of those in [the middle of an accompanist's block] just to give a break.

**Is there anything else in scheduling that you think would be helpful to increase efficiency?** I don't think so. Those are the factors you've got to take into account, and then it's just making sure that the exams then fit into the spaces that we've got. So sometimes I have to juggle around. I might have a whole batch, say, of flute exams, they're going to run either side of lunch, but I need that lunch break to begin at about 12:35 to time with our lunch break because of noise issues. So it might mean that rather than running the grades in order, I might stick a higher grade in. It's going to take a bit longer but it'll take me up to the right time. Whereas if I had two lower grades, it would overrun.

**Please could you outline the people that are involved when you're scheduling the exams. So like basically anyone that you would be informing that there are exams going on?** On who needs to know that we've got exams taking place, I would try to put it on my calendar if I can because then all the staff will know about it. Obviously first and foremost it's the heads of music [music teachers] because they're gonna have to move out of their rooms to accommodate it. So then we have to let the office know again for rescheduling classes. And caretaking staff we alert because we're trying to put things outside to keep people away, keep the noise down and we will announce it to all staff in briefing, on the days, or just before, just as a general, can you try and keep people quiet.

**Okay, And then the accompanists as well?** And yes, and once I've got a draft, what I normally do is send that to the accompanist and to the teachers in the first instance to check that it's okay from their point of view. And then, if nobody comes back to it with any major problems, then I send it out to students. So I'll print out a notice, something like that [shows me a notice], which is for accompaniments that I'll stick on the board, right? But these days, I email it around to everybody as well. I'll email it to the teachers of those pupils and the pupils themselves. And I've got the email addresses for the pupils collected as part of the entries. I've got those in my spreadsheet. Teachers emails, I've got them separately, but I have got details of who the teachers are also that could quite easily be looked up.

**I just noticed that that sheet is for practice times, right? So could you tell me more about that?** So if it's somebody like Mr Palmer who is going to come in and play for them, obviously, you're not going to go into the exam cold playing with somebody the first time. So we arrange a practice session about the week before the exam. All right? So anybody that's

paying for him to play, basically he comes in, and does rehearsal. Again, the length is determined by the grade roughly speaking. It's a little bit flexible, but that's what we should do.

All right, sounds good. What do you think is good about this whole method? It's kind of got to be done like that, I mean the spreadsheet is nice to me because I can just copy and paste things around and because it will calculate how much time we've got. [showing me spreadsheet] So if you get that drive time table there, I then cut and paste. So, for example, Mr. Meadows is playing for those ones and he was available from, I think half nine. There we go. That's his availability, Monday 9:15 to 2:15. Biggest ones needed to be Monday but after 12:45, so it had to go in the afternoon session. So, yeah, I have to work around those first of all. And sometimes, if somebody was saying they've got a couple of pupils, I almost try and put them in first if I can because if it then means I've got the rest of the morning free or something you can use that. And you can see there's Mr Palmer in the middle. Basically this extra column here is where I start with the accompanist making sure those are grouped. Okay, yes so he couldn't start early enough so I had to put some exams before. So I've stuck some guitar exams in first of all, because they don't need an accompanist. Then I've got Mr. Meadows then I've got Mr Palmer going either side of lunch which gives him a little bit of break. Mr Drew and then the rest of the day is small guitars.

So I just know it's noticed there's a payment section. So in terms of that, like when do you pay for the exam? For the exams pupils have to pay when they make the entry that comes to me, but I have to pay the board. And at the moment, I'm getting any accompanying fees separately, sort of a week later. Once I've got the basic exams done, anybody that I then know needs accompanying, I contact them separately and say, okay, it's gonna cost X amount.

And what are the current problems with this method, not just to you but to anyone that's involved? I suppose one thing is that it's not visible to anybody else until I give it to them. So ideally a system where you've got something that's produced where people could go online and find it or that automatically would send it out, would be ideal. That might be pushing it too far. It's also being able to tell me if there's problems. I mean I tell people to email me and that does generally work as long as I pick up the emails and actually act on them. But I try to get it as right as I can the first time. And usually the only thing that would then be an issue once I've published this out would be if somebody turns around to me and says, oh actually, I've got a trip that day or I've got a dental appointment. And then I have to then look at juggling things around but at that point, I'd be trying to juggle within the blocks that I've got. Say, okay, you can't do the afternoon but we'll put you at the beginning of that session. This is an example Wednesday, I've stuck a sixth and a seventh grade exam there to bring us up to 12:36. Whereas normally I probably run them in grade order going up but then that wouldn't have worked because with those lower grade times it would overrun a little bit.

All right, so then you talked about automatically putting it out, can you elaborate about that? I suppose in an ideal world and I say this is probably a step too far at this point in time. But a little bit like Practice Pal so that you get an email about when your lesson time is.

I was thinking, what if in this system, when you log on and see your time slot, what if there was an option to write a comment like 'I'm not available at this time', would that be helpful do you think? Yes, or a sort of auto reply maybe? Yeah, something like that. I'm very bad at going into systems to look at things. So it's better if I'm organising it for it to come back to me.

So what about having an email system with a little comment saying if there's any problems please email? Yeah, something that could just be a comment within the system would be quite easy.

Could you please send me the document that tells you how long each exam is? Yes I can. I'll have to find it, but I can, it's online.

Okay, Thanks. Apart from the timings and accompanists, are there any major changes between different students' exams? No. It's always the same examiner? Yes, it would be very rare to ask for two examiners on the same day. We've done it about once, mainly because of space. But once we did have two trinity examiners in, we had one in here who knows how to do percussion exams in this room. And guitars, something where you don't need a big space? Whereas we want the resonance of the room upstairs for things like flutes.

Yeah, so I just wanted to check, are the accompanists paid for these exams? Yes. So I act as a middle man, I collect the payments from parents and then I will pay it directly to accompanists.

So the parents have to pay extra to get an accompanist. Yes.

So the accompanists, do they always play the piano, or is it sometimes different? Yes, always the piano, but there are some exceptions, on the Trinity syllabus at the low grades, there are a couple of duets where a teacher could play with their student, and it would be like, you know, two guitars together or something. I don't think I've ever had anybody do any of those options. If they did, I'd kind of treat that as being the teacher as accompanying and in terms of timetabling it wouldn't make any difference.

If you're playing the piano as an accompanist to drums, but then, you're playing piano as an accompanist to a violin, is there any difference for the accompanist? No, it's the same person, they just need to be able to play with the music.

How do you tell everyone about the final schedule? There's a notice that goes on the boards in the school, and since the last year or two I will email it round but I have the information with email addresses now in my spreadsheet. Pre-COVID, we used to do everything on paper and I didn't actually have that information, which made it harder. I could do it for the boys but not for girls, for example. Okay, so now that I collect any email addresses as part of the entry form, it is easier for me to just cut and paste them and then do bulk emails. So do you manually email everyone? Yeah.

And when you're doing boys and girls school exams, is it all one set of exams because of joint departments? Yeah. There's no change, it doesn't matter if it's a boy or girl, it's the same process? Yeah, it's the same.

For the exam boards, what are the deadlines that they set for you? Okay, so for Trinity to book the exam, I have to give them at least two months notice, ideally three, and I have an online form. I just have to go in and say we want an exam and I have to give them my preferred date and an alternative date, which annoyingly doesn't send me a copy of what I put in which is problematic at times, and they're supposed to then come back to me and say, okay, that's fine. With Trinity, I then go into a portal and put the entry in by that. And then, they will issue an invoice to me to pay the fees and basically, until I pay the fees they won't do anything further, which I guess you can understand. And then, normally by 2 to 3 weeks beforehand, they will contact me with the name of an examiner and I'll have put the time table on the system and then we will just liaise by phone or whatever just to check final details from the examiner's point of view of where they're going, and what they've got to do. For ABRSM, it's a different system. They have a window when I have to book the exam. It's about a 10-day window at the beginning of each term. Once I've booked it, I can then start making the entries and I have to pay for them when I make them. So again I need to have all the information ready to add all of those entries in. In theory, I can add to it if there's time left

in the day up to about a month before. I can't timetable it on their system until 28 days before the exams begin which is actually too late from my point of view because I need to have already booked everything and tell people what's going on. But I then go in and I use their system just to upload the timetable for them to see. **During this window or 2 months before, when you're saying I want to do an exam on this day, do you need any information about who is taking it?** I don't in theory have to have information about who it is, but in reality, I need to know whether I need an examiner for one day, two days, or whatever. So I have to get the entries from people before I can make the entry. **So you ask everyone before this window or two months before?** Yeah.

**Okay, so what I'm thinking right now is that you have an electronic system where everyone can see their time slot when they log in, do you think that would be helpful?** I think it would be helpful because at least they've got one place where they know that that's what it should be.

**And who do you think would need a login to this system?** I suppose students would, the instrumental teachers would so that they can see when their students' exams. I mean, it could be a login system, it would obviously be more secure as a login system. Arguably, it's not sensitive data, so it could potentially be, you know, kind of like a web link. They know where to go to just see. You've got your options there. **What about the accompanists?** I suppose ideally yes, they could. They wouldn't have to pester me for the information.

**Right now, are you aware of any existing solutions to this problem?** No. **Okay, so Practice Pal is essentially what I'm doing, but it's for music lessons.** Yeah, Practice Pal is for music lessons and originally, it's basically just to help with the scheduling, being able to flag lessons to avoid, being able to fix lessons and so on. I mean it's developed far more than that now. But what we're looking at is really just something that will automate and ease the process of trying to work out the best schedule.

**Has anyone ever missed or been late to an exam?** Yes. **Okay, how common is that in?** It's not really, really common but people are supposed to arrive 15 minutes before the time that's shown which gives them time for them to get settled and unpacked and potentially have a quick run through with the accompanist. So if they've just played for somebody they'll run down, have a quick, quick run through before they go and play for real. But yeah, sometimes people just, How can I put it? They just don't seem to appreciate the fact that they need to be there 15 minutes before that. We occasionally have to ring across and chase people out of lessons.

**So then how has anyone ever had their exam rescheduled because of timetabling problems?** We probably have had once or twice when people really couldn't do the day that that the exam board sent through. It's quite recent that we've been able to pick the day, we used to pick a week and then they would say when it was in that week. And sometimes, somebody was away on a trip or something, then we would have to try and get them rescheduled to another centre. It's quite tricky to do. It's not impossible.

**What if they say, oh, I can do it in the morning but not in the afternoon. Is that common?** You get a few things, usually it's just one or two people. So usually you can then swap things around.

## 6.1.2 Recent examinee survey responses

Have you recently completed a music exam?	Which grade was this exam for?	When applying for the exam, were you asked for availability?	Did you have any issues with the time slot you were given? (ie. unavailable at that time)	If yes: Please explain the issue and how you fixed it.	Were you on time to the exam?	How did you find out about your exam time slot? Tick all that apply	What is your opinion on how you were told about your exam time slot? Tick all that apply.	How could the process of scheduling and taking music exams be improved? (ie. about accompanists, reminders, schedules etc.)	Have you ever missed, or been late to a music lesson?	How has Practice Pal affected your lessons? Tick all that apply
Yes	5	No, but I hope they did ask for availability	Yes, but I did nothing about it	I had dental surgery two weeks before the exam and was worried I wouldn't be able to reschedule it. However to avoid any extra hassle, I went ahead with it despite my shortened recovery time.	Yes, I was there 15 min before the exam as instructed	Music notice board	Very happy		Occasionally (once or twice a term)	I don't know what that is
Yes	3	No, but I hope they did ask for availability	No		Yes, I was there 15 minutes before the exam as instructed	Email to me, Email to my parents, Music notice board	Very happy		Occasionally (once or twice a term)	I don't know what that is
Yes	2	No	No		Yes, I was there 15 minutes before the exam as instructed	Email to me, Music teacher	Very happy		Occasionally (once or twice a term)	I don't know what that is
Yes	1	Yes	No		Yes, I was there 15 minutes before the exam as instructed	Music notice board	I would have liked an email reminder		Lots of times, I always forget	I use the Practice Pal app to check lesson times
Yes	5	No, but I hope they did ask for availability	No		Yes, I was there 15 minutes before the exam as instructed	Music notice board	I would have liked an email reminder, I don't check my emails often, so I would have liked my parents to receive a reminder email		Occasionally (once or twice a term)	I don't know what that is
Yes	1	Yes	No		Yes, I was there 15 minutes before the exam as instructed	Email to me, Music notice board	Very happy	An email reminder the day before	Never	I don't know what that is
Yes	7	No	No		Yes, I was there 15 minutes before the exam as instructed	Music notice board, Music teacher informed me about the time within the lesson	Very happy, I don't check my emails often, so I would have liked my parents to receive a reminder email	An email to both parent and student would be better, but overall the scheduling process is fine.	Occasionally (once or twice a term)	I don't know what that is
Yes	4	No, but I hope they did ask for availability	No		Yes, I was there 15 minutes before the exam as instructed	Email to me, Music notice board	Very happy		Occasionally (once or twice a term)	I don't know what that is
Yes	5	No	No		Yes, I was there 15 minutes before the exam as instructed	Music notice board	I would have liked an email reminder		Lots of times, I always forget	I don't know what that is
Yes	5	No	No		Yes, I was there 15 minutes before the exam as instructed	Music notice board, My teacher also told me.	Very happy		A few times	It is useful to get an email for when my music lesson is as it means I never forget and don't have to check on the board by the music rooms.
Yes	5	No, but I hope they did ask for availability	No		Yes, I was there 15 minutes before the exam as instructed	Music notice board	I would have liked an email reminder	I would appreciate if they sent emails out to the students to tell them about my exam, as I wasn't actually told about the date of my exam until a week before and the exam time on the music notice board was wrong		Occasionally (once or twice a term)
Yes	5	No	No		Yes, I was there 15 minutes before the exam as instructed	Music notice board	I would have liked an email reminder			Occasionally (once or twice a term)
Yes	7	No, but I hope they did ask for availability	No		Yes, I was there 15 minutes before the exam as instructed	Email to me, Music notice board	Very happy			Occasionally (once or twice a term)
Yes	5	No	No		Yes, I was there 15 minutes before the exam as instructed	Music notice board	Very happy	Emails sent with exam timing and asking for availability		Occasionally (once or twice a term)
Yes	4	No	No	Not necessarily I wasn't available but it was on a strike day so there was extra hassle and I couldn't use the day for rest or a slightly longer weekend holiday where I could have gone somewhere instead	Yes, I was there 15 minutes before the exam as instructed	Music notice board	I would have liked an email reminder, I don't check my emails often, so I would have liked my parents to receive a reminder email	First come first serve slot form, or asking when people are available and higher grades getting more priority for timings		I like getting emails for every lesson, I now remember to go to lessons more, I use the Practice Pal app to check lesson times
Yes	6	No	No		Yes, I was there 15 minutes before the exam as instructed	Music notice board	Very happy			Occasionally (once or twice a term)
Yes	3	No	Yes, but I did nothing about it	it was on a strike day and I forgot my hayfever tablets	Yes, I was there 15 minutes before the exam as instructed	Music notice board	I don't check my emails often, so I would have liked my parents to receive a reminder email	having you let a second go without taking any marks off is handy as the nerves gets increasingly 'heavy' when you're in the room		Occasionally (once or twice a term)
Yes	3	No	No		Yes, I was there 15 minutes before the exam as instructed	Email to me, Music notice board, I was told and reminded multiple times by my music teacher	Very happy	Perhaps in future students could receive notifications via edulink		i prefer having it on the music board because i'm in there often

Yes	3		No		Yes, I was there 15 minutes before the exam as instructed	Email to me, Music notice board	They got the times incorrectly at first, but i think an email is good		A few times	I don't know what that is
Yes	3	No	No		Yes, I was there 15 minutes before the exam as instructed	Email to me, Email to my parents, Music notice board	Very happy	N/A	Occasionally (once or twice a term)	I like getting emails for every lesson, I now remember to go to lessons more

## 6.2 Post-development Ms Pearcey interview transcript

■ Me

■ Ms Pearcey

**Initially, I gave Ms Pearcey an overview of the system.**

I think first of all I have some questions but you should probably just, yeah, have a look. So this is the login and registration page. Because you're not logged in yet, the dashboard will just redirect to login. And also you can register as a music teacher or an accompanist. That's in one page and the student is on another page because there's slightly different information. And you don't have to go through that process because I already kind of made you a login.

That's very kind of you.

So in real life, you would have access to this first, so you would register with a certain email and that email would be hard-coded into the system and that way you can have admin privileges. So, once you log in, this is your dashboard for you as a teacher and an accompanist as well. And the admin stuff is on another page, so here it shows, you aren't teaching anyone yet and you aren't accompanying anyone yet. So yeah, when you come onto the admin dashboard, then you can view an existing schedule or start a new one. And if you view an existing one and you've got this and the whole schedule is displayed and there's no buttons, you just view it from here.

Right, ok.

And then if you start a new one, then it'll delete the existing one. And that basically means that it'll delete some parts of the database. If you start a new schedule, you should be able to input the examinee and accompanist data, but I couldn't do that because it just got too difficult within the time constraints.

No worries. So what we can do is input it manually, if we really wanted to?

Yeah, you would have to go into the database and enter the data manually. But I have built an interface to enter the start date. And if you say it's in the past, then it has a bit of validation because it will show an error message. But if you want to schedule an exam on the first of March, you enter the date, put the rest of the data in the database, then you'll click 'schedule'. And there's a nice big button for that, which I hope is clear and direct.

Can't miss it.

Yeah. And when you click 'schedule' the function will run and produce a schedule, basically. This is currently using the Spring 2023 data you gave me. It takes you to this page, displays the date at the top and then displays the schedule. And the schedule is broken up by breaks. It calculates when the breaks are, and displays an empty row to break up the schedule a bit. That should make it a bit easier to understand.

Definitely. I do the same thing when I'm making the schedule.

Great. So the schedule is displayed, but then you can click 2 buttons from there. You can schedule it again, which just takes you back to the previous page. It deletes the existing data for the start time, so you can change that if needed. At this point, you can also change accompanist and examine your data if you want to adjust something. Like if you put in some data and someone wasn't available enough, then it would say up here, there'll be an error message. It would say their name, like 'Simon Palmer doesn't have enough time, he needs X more minutes'.

Then I would go to him in person and ask if he could play a little earlier or something. Then would I just click 'schedule again' and adjust the database?

Yeah, exactly. You click the button, go into the database and adjust his availability as needed. You would do the same if there were too many examinees, just click the button and change the database. Once you're happy, you can click the other button to publish the schedule. So this is what I developed after you gave me feedback for my system testing. You know how I asked you before, like order these 3 options by priority?

Yeah

So the most important one to you was gathering inputs. So I built the feature to input the start date. I tried doing the PDF. It was like, it was really difficult. It was much more complex than what I had designed before. So due to time constraints, I haven't done it. However, the third one was building the actual login functionality for everyone. And I have been able to do that.

Okay.

So if you view the schedule and then click publish, the schedule has been published, you get back to the admin dashboard. And then at this point, you can view it again, or start a new one. But if you were to login as a Simon Palmer, then you would see that he is accompanying these exams.

Oh that's good.

So, for Mr Palmer, all of his exams are displayed on the screen. It's just his exams, to make it really easy for him to see when everything is. And it's ordered by start time. And if you were to look at a certain student. Yeah for each student, like this guy, it would say when his exam is.

And it even says who the accompanist is and everything else. Wow, yeah I like that.

Thank you. And we can try with one more person. John Phillips. He is a teacher, and does not accompany anyone. So it has a little message 'you're not accompanying anyone' but all of his students are shown in the table there.

So you know what time your students' exams are.

Yeah, exactly. And ideally, this would also include functionality for emails as well, but I haven't been able to do that. But that's like a future thing that I could do. Because I have all of everyone's emails in the database. And this is Mr. Meadows, who's an accompanist and a teacher. In this case, it just displays it twice. And it's a bit of repeated data, which I think just makes it clear that you're teaching *and* accompanying these students. So all supervisors will see 2 tables: one for their students' exams, and one for the exams they are accompanying.

Yeah, because you could potentially have somebody playing for some of your high grades or something. I think that all seems quite sensible.

That's great. And if you were to actually delete the existing one, then it would just display for everyone this standard message. It would say nothing's scheduled yet. Also, I realised that if you have a student and they just finished their exams, what if someone logs back in, like, oh, where are my results? They might think that this page is also for results. I just want to clear any misunderstanding. So I've put a little message saying that results will be shared on the notice board. And it's similar for accompanists as well.

But yes, that's everything. This is the spring 23 data. And that's the schedule that's been produced. So you can just have a look at that. And I've also got summer and winter 23. And you can compare the ones that I made to these ones, which are the ones that you shared with me.

Okay, okay. There's only one thing that kind of jumps out at me, which is this, this exam here. Which is a bit rogue.

That's a bit of an error because of just the way that the algorithm works. It kind of recognizes at this point that Simon Palmer, like he needs more time. He has too little free time. So he needs to have a bunch of exams.

Too little free time, I love that idea. Yeah. That would be the only thing that would be problematic in real life. Because I see John would want to be in and have all of his together. The break in between accompanists is not a problem at all. And that was one of my suggestions, wasn't it? That you could pad out little gaps with guitars if you needed to.

Because in this case, what the algorithm does is that no one's available. So then it'll iterate through everyone and then it'll find someone with no accompanist and say, oh, this exam can work. And then it'll just schedule it. And then from 10am, Mr Meadows can start accompanying exams. So I'll just show you one more. So here is summer 2023. So this is the one that's created for the summer as well.

OK.

And you can just look through that.

Yeah. Yeah. I mean, that's ideal. Very good. I mean, it's always difficult kind of balancing the accompanist over the blocking instruments over time that's available. So that works very well again. John was only available from 9.30 or something, so it seems sensible.

Yeah, that's right.

So if I was doing this manually, the one change I would probably make is change the first 2 exams to guitars. So then I could have all of Mr Palmer's exams in one block. I think that would fit in there. It comes down to the breaks as well. And I know it's quite challenging getting the breaks in the right place, isn't it?

Yeah. Well, yeah, about that. So most of the breaks line up perfectly with our school lunchtime. So it's like most of them are around 12.35.

Mm-hmm.

And for this one, it's 15 minutes before this time. So it's 10.43. And this one is 3.17.

It should be about right, yeah.

Yeah, because you told me specific times for when the schedule should be. I've put them into the system. So if you did want to change them, it would be literally as easy as just changing the value in the program.

Okay, yeah.

**Now that Ms Pearcey has seen the full system, I asked her some questions.**

Okay, so I've got some questions. Does the schedule adhere to the TCL exam board regulations?

Yes, definitely.

Do you think it could have saved any more time or been more efficient?

I don't think so. Not really.

Okay. Is this how you would schedule exams?

As I say, the only difference is that I would have put all of Simon Palmer's exams together. And maybe adjusting the times a little bit on the guitar exams that are padding it out.

Yeah.

I'd have put a couple of guitars at the beginning of the day before John started. And then I'd have put all of Simon's there, which probably would have shifted that break and your algorithm might not have had the extra minute or two to do that. And it probably is just a minute or two.

And this is for all the schedules that you've seen, do you think that the accompanists would be happy with the schedule?

Essentially, yes. Yeah. I mean, it's just the hanging around. Of course, they will be happy because it meets everyone's availability. Right?

Yes. So they should be happy. And they are grouped into blocks as well.

I mean, John's are all together there. Elliot's are all together there. Simon's are mostly together. We've got these, just these two blocks. It would just mean him hanging around, wouldn't it? For what; about an hour and a quarter or something there, but sometimes that's good because you want a bit of a break because if you're doing like a whole day of exams I think it's a bit too much if you're doing them back to back.

Okay. Actually, I just want to do one more example with Winter 2023; Just that way you can see like all three examples. Yeah. So this is it.

Yeah. So that one's, that one's even better in the sense that you've, you've only got a short gap whilst he's waiting for those two exams. Again, in reality, I might have chucked a grade one guitar in there. And then, but then all of his wouldn't fit it in the morning, would they? Isn't that part of the problem?

Yeah.

Yeah. We would have spilled over. So yeah. So it was better. It is better like that.

Great.

I like the fact that the instruments are blocked in grades as well. Because from an examiner point of view, that's obviously easier too. I tend to do that because I've got my examiner hat on when I do this, actually.

Yeah.

But it doesn't matter if they're a bit more mixed up.

Right. Okay. Because that was my question. Do you think they could have been grouped any better? Because the way that I've tried to group them is within accompanist blocks, as long as the availability works.

Yeah.

And then within each accompanist block, they're grouped by family.

Good.

And then by instrument. And then they're sorted ascending by grade.

Yeah. Yeah. That's great. I mean, it's almost more than it needs to be potentially, which is not a bad thing. Yeah.

Yeah.

Most examiners also would like you for that.

Thanks. So in terms of the grouping, the main issue is the odd exam that's out of place, like in the previous one.

Yeah, but that's not too worrying.

Okay. Would you change anything about the scheduling functionality?

I don't think so.Um, I mean, what you might want to put as a future thing you could think about if, you know, you were developing this commercially, et cetera, et cetera, is maybe once you've got this stage of having an option to sort of manually swap 2 students, for example. So I might get to the stage and then somebody will say to me, well, actually, I can't do that time because of this and that, and you know, you sometimes get extra restrictions put in. So if at that point you can take this nice display and just make manual changes.That would be ideal.

The idea was that you would have an edit button down here and then it would be some kind of drag and drop feature.

Yes I remember. But that would be quite hard.

Yeah, it would take too much time. And I tried not to do other things like that until the main scheduling feature was done, you know?

Definitely, it would be a lot less useful if the key scheduling capabilities were bad. I think you've done a good job with developing what needs to be developed. Um, but this would be more of an ideal extra.

Okay. How much time do you have?

Uh, it's quarter past. So 10 minutes.

Okay. I'll be quick. Um, that's, so yeah, um, I do want to compare this one to yours. This is the Winter 2023 data. So if we just compare the winter to the one that you created, how would you, like, how would you compare it?

Okay. So we have the saxophones for the same beginning. Um, okay. So I suppose the difference is that I've got all of Simon's together and then I've got John's, but it's possible that I gave you slightly different availability times in the information I gave you. Yeah. Yeah. So and then the afternoon is going to be pretty much the same, I'd imagine because you've got a similar- so the reason that that grade four is up there is because she couldn't go later than when originally scheduled.

Why is this, like, mixed up over here?

Different teachers, actually. So I had grouped Mr. Phillips together and Miss Palmer's, I think it was. It's not important for them, because they don't come along. Um, I suppose if you had a teacher that you knew was going to come along to be there to support students beforehand, then it would be another factor to consider.

Right. So they're grouped by teacher because teachers might want to support their students. Yeah.

And just, I suppose my spreadsheet in the first place, that tends to be how I get organised. So when I bring them across, I do it the same way.

So, would you say this is better or worse than the one that you created?

I think the only thing that I think is worse is that little bit there where John's is the middle of Simon's. Um, but that might be down to the availability I told you he had.

Okay.

Um, and I think so. I said in the email, sometimes you have to kind of go back and renegotiate. Yeah, when I do the times. So they'll say, well, I'm available between 10 and 11. And I'll email them, I see you've got an hour and three quarters worth of exam. So in which case I'll ask if they can stay later.

I just, I'm interested because there's one, there's one where, um, availability for, yeah. So he, he was, um, 9:30 for me. So, this one is 9:26. So I guess for you, you just say, can you come in four minutes earlier?

Yeah.

Because the system was strict. So do you think I could change it so that whatever availability they say, you just add five minutes each end.

Um, that would be an idea to sort of have a sort of contingency bit. And then they say, yeah, maybe a way of sort of highlighting it that it just says, well, actually this is slightly earlier than they said they could or slightly later.

Right, okay. But I think, I guess if you could just edit manually, then it would just be solved.

If you got to the stage where you could just edit it, that would, that would get around that.

Yeah. Okay. So, um, that's everything for the scheduling stuff. I'll log in as a supervisor just to show this thing. Um, actually in the meantime, do you think the interface is simple and intuitive?

Yes, I do. It's very, very well made.

Thank you.

Um, would you want, because you know, I have included these little messages and little tips as you're doing things. For example, this warning is only displayed if there's an existing schedule already to prevent confusion. So do you think these little messages are helpful?

Yes, they are quite helpful.

Is there anything else you would want from these messages?

I don't think so.

Okay. So that's ready now. So, for an accompanist like Mr Palmer, he'll login and see what his times are. Do you think it's accurate and it's relevant to just him?

Yeah, definitely.

And do you, do you think that's like, do you think that's a good thing? Do you think it's better for them to see only their exams or better for them to see everyone's exams?

Probably better to see what they need to see. Yeah. Yeah. So this is good. This is fine. Yeah.

Okay. Then, um, are you happy that only you can access the scheduling functionality and other admin features?

Yes, I am.

Cause, you know, for example, with Mr. Palmer.

Its security. Not letting anybody else get to the directory, and fiddle with it.

Exactly. Yeah. There's no way for them to access all that stuff.

Yeah.

The same with a student. If you just display the one timeslot, or if you're a teacher and just display their student's exams, that's all fine?

Yeah, it's perfect.

Okay.

Perfect.

And, okay. So this is the last bit. So about the scheduling results that you see on the screen. Are they easy to understand?

Yes. Very easy.

And would you want any other data to be displayed?

Not that I can think of.

Right. Actually, that being said, I have added the teacher name just for clarity. Because when a teacher is seeing the schedule, I thought it would be better for them to see their own name there.

Yeah, that's a good idea.

Um, and the schedule button, it's kind of inspired by Practice Pal, and their magic scheduling.

Yes.

Are you happy that this button is easy to press, and simple?

It's very easy and clear. Easy to press. Yeah.

Very nice. And the same goes for the publish schedule button.

That's, yeah. Yeah. No, it's very clear.

For the interface itself, do you think it's kind of intuitive to understand, and not just for you, but for accompanists and students of all different ages and capabilities?

Yeah. So the schedule has been made. If I press schedule again, does that restart the whole scheduling process?

It takes you back to the admin dashboard and it doesn't delete anything here because it's kind of just giving you control over that. It just deletes the start date.

Right. Okay. Yeah.

In the future, when I build the interface for uploading a file for examinees and accompanists, it would delete that stuff, or there would probably be some options, like what do you want to get rid of? But yeah, click schedule again, it'll take you back here and you just do it again.

Click your date. Yeah.

At that point you would change the database stuff if you wanted to.

Okay.

Would you change anything to make the system more usable?

The only thing that might be nice would be to have it kind of save that previous schedule somewhere so that if you did accidentally delete it, and not that it takes a long time to click the buttons again, but, so that you could see it or you could compare them potentially.

So like you can maybe load an existing schedule and go from there.

Yeah.

Right. And is there any unnecessary information displayed on the screen at any point that includes when the schedules are displayed, is there anything that you think like there's no need for that to be displayed?

No.

That's good to hear.

I like the calendar thing to select the date.

Thanks.

We get very used to doing it, don't we? But it does make a difference.

Yeah.

And I presume the way that your algorithm is written, it's going to give me the same results each time.

Yeah. It's one to one.

Yeah, good.

Actually, can I just show you if we have time. Do you have to go?

No, I'm all right for a minute.

Okay.

I just need to go and get a drink in a minute.

So this is the Winter 2023 data. So if we go into winter. If you were to change Mr Palmer's availability. Let's just say that he's not available for the morning. Then it could return an error. And then you would click schedule again, and the schedule button. Okay. So he's got less availability, so it's scheduled him in a later block.

Yeah.

What about if we do it even more?

Okay.

Now it should return an error because he's playing too many exams. He isn't available enough. Yeah. So it says Simon Palmer has too many exams. He needs at least 84 minutes or more.

Yeah.

And I've emphasised the 'at least'.

Yes.

Yeah. And yeah. So it does display the schedule up until it finds the error.

Yeah.

So it'll just keep going. And that's true if you have multiple accompanists that are like this, or if there's any other error that can come, it'll just display on screen.

Yeah.

And because you, very kindly, cut the data down to fit a day. But if you did add another examinee, it would say the day has ended and these guys haven't been scheduled. Um, and yeah. So what do you think about that?

That's, again, very good. Yes.

Okay.

Yeah. Yes. I mean, when it says they haven't been scheduled, if it says how much time they need, just by adding together their exam times. That would potentially be useful if you're thinking about going to have to book another day or half day, then you know how much you've got to book.

Okay.

It's not strictly necessary.

No, that's definitely a good point.

It's just another thought really. You know, it's a bit like you said it needs, he needs at least 84 minutes. I mean, In theory, you could add up all the minutes it's going for, you know, subtract what has been scheduled and then you know how much time you need.

And you see at, at the bottom of this, you have this message to email you, and all these little messages just to kind of help.

Yeah, it seems well thought out.

OK. That's it.

I am really impressed.

That's sound, bro.

SoundBro. I like the name.

Thank you.

## 6.3 References

These references were used to identify the TCL regulations. Rules can change, so my system is designed for these rules as of August 2023.

Name	Link
Duration of each exam, and all TCL regulations	<a href="http://www.trinitycollege.co.uk/MusicTimetabling">http://www.trinitycollege.co.uk/MusicTimetabling</a>
All TCL Classical and Jazz instruments, grouped by instrument family	<a href="https://www.trinitycollege.com/qualifications/music/music-certificate-exams">https://www.trinitycollege.com/qualifications/music/music-certificate-exams</a>
All TCL Rock and Pop instruments	<a href="https://www.trinMusic Certificate exams   Trinity College Londonityrock.com/exams/syllabus">https://www.trinMusic Certificate exams   Trinity College Londonityrock.com/exams/syllabus</a>
General information and guidelines on the scheduling process	<a href="https://www.trinitycollege.com/resource/?id=3980">https://www.trinitycollege.com/resource/?id=3980</a>

## 6.4 Final code

The final structure of the directory is shown below:

```
soundbro/
|--- index.php
|--- login.php
|--- register.php
|--- register-student.php
|--- register-supervisor.php
|--- dashboard.php
|--- dashboard-admin.php
|--- schedule.php
|--- schedule-complete.php
|--- logout.php
|--- styles.css
|--- includes/
    |--- db-connect.php
    |--- navbar.php
    |--- footer.html
|--- logo/
    |--- full.png
    |--- favicon3.png
```

All code in these files have been displayed below.

## INDEX.PHP



```
1  <?php
2  session_start();
3  ?>
4  <!DOCTYPE html>
5  <html lang="en">
6
7
8  <head>
9      <meta charset="UTF-8">
10     <meta name="viewport" content="width=device-width, initial-scale=1,
shrink-to-fit=no">
11     <!-- connect to stylesheets -->
12     <link href='https://fonts.googleapis.com/css?family=League Spartan'
rel='stylesheet'>
13     <link rel="stylesheet" href="https://cdn.jsdelivr.net/npm/bootstrap-
icons@1.11.2/font/bootstrap-icons.min.css">
14     <link href="https://cdn.jsdelivr.net/npm/bootstrap@5.3.2/dist/css/b
ootstrap.min.css" rel="stylesheet" integrity="sha384-T3c6CoIi6uLrA9TneN
Eoa7RxnatzjcDSCmG1MxxSR1GAsXEV/DwykC2MPK8M2HN" crossorigin="anonymou
s">
15     <link href='styles.css' rel='stylesheet'>
16     <link rel="icon" href="logo/favicon3.png">
17     <title>SoundBro</title>
18 </head>
19
20 <body class="bg-light">
21
22 <!-- display navigation bar -->
23 <?php include 'includes/navbar.php' ?>
24
25 <!-- welcome message + image -->
26 <div class='h2 p-2 m-5 text-center'>Welcome to <b class="logo fw-bolde
r fs-2 ">SoundBro</b>! Follow the links above to get started.</div>
27 
28
29 <!-- display footer -->
30 <?php include 'includes/footer.html' ?>
31
32 </body>
33 </html>
```

## LOGIN.PHP



```
1  <?php
2  // Start the session to store user data
3  session_start();
4
5  // If logged in, redirect to the dashboard
6  if (isset($_SESSION['user_id'])) {
7      header("Location: dashboard.php");
8      exit();
9  }
10
11 // connect to database
12 include 'includes/db-connect.php';
13
14
15 // if form is submitted
16 if ($_SERVER['REQUEST_METHOD'] === 'POST') {
17
18     // Retrieve inputs + prevent sql injection
19     $email = mysqli_real_escape_string($conn,$_POST['email']);
20     $password = mysqli_real_escape_string($conn,$_POST['password']);
21
22     // validation - presence check
23     if (!$email or !$password){
24         $error="You must enter a username and password";
25     } else {
26
27         // Validate email
28         $valid= filter_var($email, FILTER_VALIDATE_EMAIL);
29         if (!$valid) {
30             $error = "Invalid email format";
31         } else {
32
33             // Fetch login info for both students and supervisors
34             $studentQuery = "SELECT StudentID,Password FROM Students WHERE Email
1 = '$email'";
35             $supervisorQuery = "SELECT SupervisorID,Password FROM Supervisors W
HERE Email = '$email'";
36             $studentResult = $conn->query($studentQuery);
37             $supervisorResult = $conn->query($supervisorQuery);
38
39             if (!$studentResult or !$supervisorResult) {
40                 $error = "Could not retrieve database info. Try again later.";
41             } else {
42
43                 // Check if email exists in the students table
44                 if ($studentResult->num_rows > 0) {
45
46                     // student email found
47                     $row = $studentResult->fetch_assoc();
48                     $passwordDB=$row['Password'];
49
50                     // Verify the password
```



```
50          // Verify the password
51      if (password_verify($password,$passwordDB)) {
52
53          // Password is correct, set session variables
54          $id=$row['StudentID'];
55          $_SESSION['user_type']='student';
56          $_SESSION['user_id'] = $id;
57
58          // Redirect to dashboard
59          header('Location: dashboard.php');
60          exit();
61      } else {
62          // student password is incorrect
63          $error = "Incorrect password";
64      }
65
66  } else {
67
68      // check if email exists in the supervisors table
69      if ($supervisorResult->num_rows > 0) {
70
71          // supervisor email found
72          $row = $supervisorResult->fetch_assoc();
73          $passwordDB=$row['Password'];
74
75          // Verify the password
76          if (password_verify($password,$passwordDB)) {
77              // Password is correct, set session variables
78              $id=$row['SupervisorID'];
79              $_SESSION['user_id'] = $id;
80
81              if ($email=='admin@email.com') {
82                  // admin login (Ms Pearcey)
83                  $_SESSION['user_type']='admin';
84              } else {
85                  $_SESSION['user_type']='supervisor';
86              }
87
88              // Redirect to dashboard
89              header('Location: dashboard.php');
90              exit();
91          } else {
92              //supervisor password is incorrect
93              $error = "Incorrect password";
94          }
95      } else {
96          $error = "Email not found";
97      }
98  }
99 }
100 }
101 }
102 }
```



```
103 // Close the database connection
104 $conn->close();
105 ?>
107
108 <!DOCTYPE html>
109 <html lang="en">
110
111 <head>
112     <meta charset="UTF-8">
113     <meta name="viewport" content="width=device-width, initial-scale=1, shrink-to-fit=no">
114         <!-- connect to stylesheets -->
115         <link href='https://fonts.googleapis.com/css?family=League Spartan' rel='stylesheet'>
116         <link rel="stylesheet" href="https://cdn.jsdelivr.net/npm/bootstrap-icons@1.11.2/font/bootstrap-icons.min.css">
117         <link href="https://cdn.jsdelivr.net/npm/bootstrap@5.3.2/dist/css/bootstrap.min.css" rel="stylesheet" integrity="sha384-T3c6CoIi6uLrA9TheNEoa7RxnatzjcDSCmG1MXxSR1GAsXEV/DwykC2MPK8M2HN" crossorigin="anonymous">
118             <link href='styles.css' rel='stylesheet'>
119             <link rel="icon" href="logo/favicon3.png">
120             <title>Login-SoundBro</title>
121         </head>
122
123 <body class="bg-light">
124
125     <!-- link to navigation bar -->
126     <?php include 'includes/navbar.php' ?>
127
128     <!-- login container -->
129     <div class="container my-5">
130         <div class="row justify-content-center">
131             <div class="col-md-6 bg-white p-5 rounded-4 shadow">
132                 <h2 class="text-center mb-4 text-primary">Login</h2>
133
134                 <!-- display if user has just registered -->
135                 <?php if (isset($_GET['registered']) and $_GET['registered']==1) { ?>
136
137                     <div class="alert alert-success">
138                         Registration successful!
139                     </div>
140                 <?php } ?>
141
142                 <!-- display if error is found -->
143                 <?php if (isset($error)) { ?>
144                     <div class="alert alert-danger">
145                         <?php echo $error; ?>
146                     </div>
147                 <?php } ?>
148
149                 <form method="post" action="">
150                     <!-- enter email -->
151                     <div class="form-group">
152                         <label for="email">Email address</label>
153                         <input type="email" class="form-control" id="email" name="email" >
154                     </div>
```



```
155
156      <!-- enter password -->
157      <div class="form-group">
158          <label for="password">Password</label>
159          <input type="password" class="form-control" id="password" na
160              me="password">
161
162          <!-- submit -->
163          <div class="mt-2 form-group">
164              <button type="submit" class="btn btn-primary btn-block">Logi
165          n</button>
166      </div>
167
168          <!-- link to registration page to prevent confusion -->
169          <p class="mt-3 text-center">
170              Don't have an account? <a href="register.php">Register here</a>
171          </p>
172      </div>
173  </div>
174 </div>
175
176 <!-- display footer -->
177 <?php include 'includes/footer.html'?>
178 </body>
179 </html>
```

## REGISTER.PHP

```
 1 <?php
 2 session_start();
 3 // if logged in redirect to dashboard.php
 4 if (isset($_SESSION['user_id'])) {
 5     header("Location: dashboard.php");
 6     exit();
 7 }
 8 ?>
 9
10 <!DOCTYPE html>
11 <html lang="en">
12
13 <head>
14     <meta charset="UTF-8">
15     <meta name="viewport" content="width=device-width, initial-scale=1, shrink-to-fit=no">
16     <!-- connect to stylesheets -->
17     <link href='https://fonts.googleapis.com/css?family=League Spartan' rel='stylesheet'>
18     <link rel="stylesheet" href="https://cdn.jsdelivr.net/npm/bootstrap-icons@1.1.2/font/bootstrap-icons.min.css">
19     <link href="https://cdn.jsdelivr.net/npm/bootstrap@5.3.2/dist/css/bootstrap.min.css" rel="stylesheet" integrity="sha384-T3c6CoIi6uLrA9TneNEoa7RxnatzjcDSCmG1MXxSR1GAsXEV/DwykyC2MPK8M2HN" crossorigin="anonymous">
20     <link href='styles.css' rel='stylesheet'>
21     <link rel="icon" href="logo/favicon3.png">
22     <title>Registration - SoundBro</title>
23 </head>
24
25 <body class="bg-light">
26 <!-- display navigation bar -->
27 <?php include "includes/navbar.php" ?>
28
29 <div class="container my-5">
30     <div class="row justify-content-center">
31         <div class="col-md-8 bg-white p-5 rounded-4 shadow">
32             <!-- direct users to correct registration page -->
33             <h2 class="text-center mb-4 text-primary">Register</h2>
34             <h5 class="text-center mb-4">To register a student, click <a href='register-student.php'>here</a></h5>
35             <h5 class="text-center mb-4">To register an accompanist / music teacher, click <a href='register-supervisor.php'>here</a></h5>
36             <p class="h6 mt-3 text-center">
37                 <!-- link to login page -->
38                 If you have an account, login <a href="login.php">here</a>
39             </p>
40         </div>
41     </div>
42 </div>
43
44 <!-- display footer -->
45 <?php include 'includes/footer.html'?>
46 </body>
47
48 </html>
```

## REGISTER-STUDENT.PHP

```
1 <?php
2 session_start();
3
4 // if logged in redirect to dashboard.php
5 if (isset($_SESSION['user_id'])) {
6     header("Location: dashboard.php"); // Redirect to dashboard
7     exit();
8 }
9
10 // connect to database
11 include 'includes/db-connect.php';
12
13 // if form is submitted
14 if ($_SERVER['REQUEST_METHOD'] === 'POST') {
15
16     // Retrieve inputs + prevent sql injection
17     $firstName=mysqli_real_escape_string($conn,$_POST['firstName']);
18     $lastName=mysqli_real_escape_string($conn,$_POST['lastName']);
19     $email = mysqli_real_escape_string($conn,$_POST['email']);
20     $parentEmail = mysqli_real_escape_string($conn,$_POST['parentEmail']);
21     $password = mysqli_real_escape_string($conn,$_POST['password']);
22     $confirmPassword = mysqli_real_escape_string($conn,$_POST['confirmPassword']);
23
24     // --- VALIDATION ---
25     // presence check
26     if (!$firstName or !$lastName or !$email or !$parentEmail or !$password or !$confirmPassword) {
27         $error="You must enter data for all fields";
28
29     } else {
30
31         // check first name validity - Name must be 2-50 chars
32         $validLength_fname = strlen($firstName) >= 2 && strlen($firstName) <= 50;
33         // no special chars (only lowercase, uppercase, hyphens, and spaces)
34         $characterSetValid_fname = preg_match("/^([a-zA-Z- ])+$/", $firstName);
35         // No numbers
36         $noNumbers_fname = !preg_match("/[0-9]/", $firstName);
37
38         if ($validLength_fname and $characterSetValid_fname and $noNumbers_fname) {
39
40             // check last name validity - Name must be 2-50 chars
41             $validLength_lname = strlen($lastName) >= 2 && strlen($lastName) <= 50;
42             // no special chars (only lowercase, uppercase, hyphens, and spaces)
43             $characterSetValid_lname = preg_match("/^([a-zA-Z- ])+$/", $lastName);
44             // No numbers
45             $noNumbers_lname = !preg_match("/[0-9]/", $lastName);
46
47             if ($validLength_lname and $characterSetValid_lname and $noNumbers_lname){
48                 //check email validity
49                 $email=filter_var($email,FILTER_VALIDATE_EMAIL);
50                 $parentEmail=filter_var($parentEmail,FILTER_VALIDATE_EMAIL);
51                 if (!$email or !$parentEmail) {
52                     $error="Invalid email / parent email";
53                 } else {
54
```



```
55 //check if user already has a login
56 $checkStudents="SELECT * FROM Students WHERE Email = '$email'";
57 $checkSupervisors="SELECT * FROM Supervisors WHERE Email = '$email'";
58 $studentResult=$conn->query($checkStudents);
59 $supervisorResult=$conn->query($checkSupervisors);
60
61 if (!$studentResult or !$supervisorResult) {
62     //handle unexpected error
63     $error = "Could not retrieve database info. Try again later.";
64 } else {
65
66     if ($studentResult->num_rows>0) {
67         // email found in Students table
68         $error = "Email already registered as a student. Please log in";
69     } else {
70
71         if($supervisorResult->num_rows>0) {
72             // email found in Supervisors table
73             $error = "Email already registered as a music teacher / accompanis
t. Please log in";
74         } else {
75
76             // check passwords match
77             if ($password !== $confirmPassword) {
78                 $error='Passwords do not match';
79             } else {
80
81                 //check password constraints
82                 $uppercase = preg_match('@[A-Z]@', $password);
83                 $lowercase = preg_match('@[a-z]@', $password);
84                 $number = preg_match('@[0-9]@', $password);
85                 if (!$uppercase or !$lowercase or !$number or strlen($password)
< 8) {
86                     $error="The password must be at least 8 characters long and
include at least one
87                     uppercase letter, one lowercase letter, and one number.";
88                 } else {
89
90                     // hash password for security
91                     $hashedPassword=password_hash($password,PASSWORD_DEFAULT);
92                     // insert into database
93                     $sql="INSERT INTO Students (FirstName,LastName,Email,Passwo
rd,ParentEmail)
94                     VALUES ('$firstName','$lastName','$email','$hashedPasswor
d','$parentEmail')";
95                     if ($conn->query($sql) === TRUE){
96                         // redirect to login page with registration message
97                         header("Location: login.php?registered=1");
98                         exit();
99                     } else {
100                         $error= "Error with database: Try again later";
101                     }
102                 }
103             }
```

```

103
104
105
106
107
108 } else {
109     //invalid last name
110     $error="Both names must be 2-50 characters, with no numbers or special characters
(except hyphens)";
111
112 } else {
113     // invalid first name
114     $error="Both names must be 2-50 characters, with no numbers or special characters (exe
pt hyphens)";
115
116
117 }
118 }
119
120 ?>
121
122 <!DOCTYPE html>
123 <html lang="en">
124
125 <head>
126     <meta charset="UTF-8">
127     <meta name="viewport" content="width=device-width, initial-scale=1, shrink-to-fit=no">
128     <!-- connect to stylesheets -->
129     <link href='https://fonts.googleapis.com/css?family=League Spartan' rel='stylesheet'>
130     <link rel="stylesheet" href="https://cdn.jsdelivr.net/npm/bootstrap-icons@1.11.2/font/bootstrap
-icons.min.css">
131     <link href="https://cdn.jsdelivr.net/npm/bootstrap@5.3.2/dist/css/bootstrap.min.css" rel="style
sheet" integrity="sha384-T3c6CoIi6uLrA9TneNEoa7RxnatzjcDSCmG1MXxSR1GAsXEV/Dwwykc2MPK8M2HN" crossori
gin="anonymous">
132     <link href='styles.css' rel='stylesheet'>
133     <link rel="icon" href="logo/favicon3.png">
134     <title>Registration - SoundBro</title>
135 </head>
136
137 <body class="bg-light">
138     <!-- display navigation bar -->
139     <?php include "includes/navbar.php" ?>
140
141     <div class="container my-5">
142         <div class="row justify-content-center">
143             <div class="col-md-8 bg-white p-5 rounded-4 shadow">
144                 <h2 class="text-center mb-4 text-primary">Register a Student</h2>
145
146                 <!-- display error if found -->
147                 <?php if (isset($error)) { ?>
148                     <div class="alert alert-danger">
149                         <?php echo $error; ?>
150                     </div>
151                 <?php } ?>

```



```
152
153     <form method="post" action="">
154         <!-- enter first name -->
155         <div class="form-group">
156             <label for="firstName">First name</label>
157             <input type="text" class="form-control" id="firstName" name="firstName">
158             <small class="text-muted">
159                 Names must be 2-50 characters, with no numbers or special characters (e
xcept hyphens).
160             </small>
161         </div>
162
163         <!-- enter last name -->
164         <div class="form-group">
165             <label for="lastName">Last name</label>
166             <input type="text" class="form-control" id="lastName" name="lastName">
167         </div>
168
169         <!-- enter email-->
170         <div class="form-group">
171             <label for="email">Email address</label>
172             <input type="email" class="form-control" id="email" name="email">
173         </div>
174
175         <!-- enter parent email -->
176         <div class="form-group">
177             <label for="parentEmail">Parent email address </label>
178             <input type="email" class="form-control" id="parentEmail" name="parentEmai
l">
179             <small class="text-muted">
180                 We'll only send you and your parent an email when the schedule is creat
ed.
181             </small>
182         </div>
183
184         <!-- enter password -->
185         <div class="form-group">
186             <label for="password">Password</label>
187             <input type="password" class="form-control" id="password" name="password">
188             <small class="text-muted">
189                 Password must be at least 8 characters long and include at least one up
percase letter, one lowercase letter and one number.
190             </small>
191         </div>
192
193         <!-- confirm password -->
194         <div class="form-group">
195             <label for="confirmPassword">Confirm Password</label>
196             <input type="password" class="form-control" id="confirmPassword" name="conf
irmPassword">
197         </div>
198
199         <!-- submit -->
200         <div class="mt-2 form-group">
201             <button type="submit" class="btn btn-primary btn-block">Register</button>
202         </div>
203     </form>
204
205
```

●

●

●

```
204
205      <!-- link back to login page -->
206      <p class="h6 mt-3 text-center">
207          If you have an account, login <a href="login.php">here</a>
208      </p>
209      </div>
210  </div>
211 </div>
212
213 <!-- display footer -->
214 <?php include 'includes/footer.html'?>
215 </body>
216
217 </html>
```

## REGISTER-SUPERVISOR.PHP

```
1 <?php
2 session_start();
3
4 // if logged in redirect to dashboard.php
5 if (isset($_SESSION['user_id'])) {
6     header("Location: dashboard.php");
7     exit();
8 }
9
10 // connect to database
11 include 'includes/db-connect.php';
12
13 // if form is submitted
14 if ($_SERVER['REQUEST_METHOD'] === 'POST') {
15
16     // Retrieve inputs + prevent sql injection
17     $firstName=mysqli_real_escape_string($conn,$_POST['firstName']);
18     $lastName=mysqli_real_escape_string($conn,$_POST['lastName']);
19     $email = mysqli_real_escape_string($conn,$_POST['email']);
20     $password = mysqli_real_escape_string($conn,$_POST['password']);
21     $confirmPassword = mysqli_real_escape_string($conn,$_POST['confirmPassword']);
22
23     // --- VALIDATION ---
24     // presence check
25
26     if (!$firstName or !$lastName or !$email or !$password or !$confirmPassword) {
27         $error="You must enter data for all fields";
28
29     } else {
30
31         // check first name validity - Name must be 2-50 chars
32         $validLength_fname = strlen($firstName) >= 2 && strlen($firstName) <= 50;
33         // no special chars (only lowercase, uppercase, hyphens, and spaces)
34         $characterSetValid_fname = preg_match("/^([a-zA-Z\-\ ])+$/", $firstName);
35         // No numbers
36         $noNumbers_fname = !preg_match("/[0-9]/", $firstName);
37
38         if ($validLength_fname and $characterSetValid_fname and $noNumbers_fname) {
39
40             // check last name validity - Name must be 2-50 chars
41             $validLength_lname = strlen($lastName) >= 2 && strlen($lastName) <= 50;
42             // no special chars (only lowercase, uppercase, hyphens, and spaces)
43             $characterSetValid_lname = preg_match("/^([a-zA-Z\-\ ])+$/", $lastName);
44             // No numbers
45             $noNumbers_lname = !preg_match("/[0-9]/", $lastName);
46
47             if ($validLength_lname and $characterSetValid_lname and $noNumbers_lname){
48                 //check email validity
49                 $email=filter_var($email,FILTER_VALIDATE_EMAIL);
50                 if (!$email) {
51                     $error="Invalid email";
52                 } else {
53
```



```
54 //check if user already has a login
55 $checkStudents="SELECT * FROM Students WHERE Email = '$email'";
56 $checkSupervisors="SELECT * FROM Supervisors WHERE Email = '$email'";
57 $studentResult=$conn->query($checkStudents);
58 $supervisorResult=$conn->query($checkSupervisors);
59
60 if (!$studentResult or !$supervisorResult) {
61     //handle unexpected error
62     $error = "Could not retrieve database info. Try again later.";
63 } else {
64
65     if ($studentResult->num_rows>0) {
66         // email found in Students table
67         $error = "Email already registered as a student. Please log in";
68     } else {
69
70         if($supervisorResult->num_rows>0) {
71             // email found in Supervisors table
72             $error = "Email already registered as a music teacher / accompanis
t. Please log in";
73         } else {
74
75             // check passwords match
76             if ($password !== $confirmPassword) {
77                 $error='Passwords do not match';
78             } else {
79
80                 //check password constraints
81                 $uppercase = preg_match('@[A-Z]@', $password);
82                 $lowercase = preg_match('@[a-z]@', $password);
83                 $number = preg_match('@[0-9]@', $password);
84                 if (!$uppercase or !$lowercase or !$number or strlen($password)
< 8) {
85                     $error="The password must be at least 8 characters long and
include at least one
86                         uppercase letter, one lowercase letter, and one number.";
87                 } else {
88
89                     // hash password for security
90                     $hashedPassword=password_hash($password,PASSWORD_DEFAULT);
91                     // insert into database
92                     $sql="INSERT INTO Supervisors (FirstName,LastName,Email,Pas
sword)
93                         VALUES ('$firstName','$lastName','$email','$hashedPasswor
d')";
94                     if ($conn->query($sql) === TRUE){
95                         // redirect to login page with registration message
96                         header("Location: login.php?registered=1");
97                         exit();
98                     } else {
99                         $error= "Error with database: Try again later";
100                     }
101                 }
102             }
103         }
104     }
105 }
```

```
 106          }
 107      } else {
 108          //invalid last name
 109          $error="Both names must be 2-50 characters, with no numbers or special characters
(except hyphens)";
 110      }
 111  } else {
 112      //invalid first name
 113      $error="Both names must be 2-50 characters, with no numbers or special characters (exce
pt hyphens)";
 114  }
 115 }
 116 }
 117 }
 118 ?>
 119
 120 <!DOCTYPE html>
 121 <html lang="en">
 122
 123 <head>
 124     <meta charset="UTF-8">
 125     <meta name="viewport" content="width=device-width, initial-scale=1, shrink-to-fit=no">
 126     <!-- connect to stylesheets -->
 127     <link href='https://fonts.googleapis.com/css?family=League Spartan' rel='stylesheet'>
 128     <link rel="stylesheet" href="https://cdn.jsdelivr.net/npm/bootstrap-icons@1.11.2/font/bootstrap
-icons.min.css">
 129     <link href="https://cdn.jsdelivr.net/npm/bootstrap@5.3.2/dist/css/bootstrap.min.css" rel="style
sheet"
 130         integrity="sha384-T3c6CoIi6uLrA9TneNEoa7RxnatzjcDSCmG1MXxSR1GAsXEV/Dwykc2MPK8M2HN" crossorigin
="anonymous">
 131     <link href='styles.css' rel='stylesheet'>
 132     <link rel="icon" href="logo/favicon3.png">
 133     <title>Registration - SoundBro</title>
 134 </head>
 135
 136 <body class="bg-light">
 137     <!-- display navigation bar -->
 138     <?php include "includes/navbar.php" ?>
 139
 140     <div class="container my-5">
 141         <div class="row justify-content-center">
 142             <div class="col-md-8 bg-white p-5 rounded-4 shadow">
 143                 <h2 class="text-center mb-4 text-primary">Register a Music Teacher / Accompanist</h2>
 144
 145                 <!-- display error if found -->
 146                 <?php if (isset($error)) { ?>
 147                     <div class="alert alert-danger">
 148                         <?php echo $error; ?>
 149                     </div>
 150                 <?php } ?>
 151
 152                 <form method="post" action="">
 153                     <!-- enter first name -->
 154                     <div class="form-group">
 155                         <label for="firstName">First name</label>
 156                         <input type="text" class="form-control" id="firstName" name="firstName">
 157                         <small class="text-muted">
 158                             Names must be 2-50 characters, with no numbers or special characters (excep
t hyphens).
 159                         </small>
 160                     </div>
```



```
161
162     <!-- enter last name -->
163     <div class="form-group">
164         <label for="lastName">Last name</label>
165         <input type="text" class="form-control" id="lastName" name="lastName">
166     </div>
167
168     <!-- enter email-->
169     <div class="form-group">
170         <label for="email">Email address</label>
171         <input type="email" class="form-control" id="email" name="email">
172         <small class="text-muted">
173             We'll only send you an email when the schedule is created.
174         </small>
175     </div>
176
177     <!-- enter password -->
178     <div class="form-group">
179         <label for="password">Password</label>
180         <input type="password" class="form-control" id="password" name="password">
181         <small class="text-muted">
182             Password must be at least 8 characters long and include at least one uppercase letter, one lowercase letter and one number.
183         </small>
184     </div>
185
186     <!-- confirm password -->
187     <div class="form-group">
188         <label for="confirmPassword">Confirm Password</label>
189         <input type="password" class="form-control" id="confirmPassword" name="confirmP
      assword">
190     </div>
191
192     <!-- submit -->
193     <div class="mt-2 form-group">
194         <button type="submit" class="btn btn-primary btn-block">Register</button>
195     </div>
196 </form>
197
198     <!-- link back to login page -->
199     <p class="h6 mt-3 text-center">
200         If you have an account, login <a href="login.php">here</a>
201     </p>
202     </div>
203 </div>
204 </div>
205
206     <!-- display footer -->
207     <?php include 'includes/footer.html'?>
208 </body>
209
210 </html>
```

## DASHBOARD.PHP

```
● ● ●
1 <?php
2 session_start();
3 // if logged out redirect to login.php
4 if (!isset($_SESSION['user_id'])) {
5     header("Location: login.php");
6     exit();
7 }
8
9 //connect to database
10 include 'includes/db-connect.php';
11
12 // get user's first name
13 $ID=$_SESSION['user_id'];
14 if ($_SESSION['user_type']=='student'){
15     $result=$conn->query("SELECT FirstName FROM Students WHERE StudentID = '$ID'");
16 } else {
17     $result=$conn->query("SELECT FirstName FROM Supervisors WHERE SupervisorID = '$ID'");
18 }
19 $row = $result->fetch_assoc();
20 $firstName=$row['FirstName'];
21 $message="";
22
23 // display the exam schedule
24 function displaySchedule($conn,$result,$teacher){
25     while ($row = $result->fetch_assoc()) {
26
27         //get start time in correct form
28         $StartTime=substr($row['StartTime'],0,5);
29
30         // display info on table
31         echo "<tr>";
32         echo "<td>{$row['student_fname']} {$row['student_lname']}
```

```
 52         //display start time
 53         echo "<td><strong>{$StartTime}</strong></td>";
 54         echo "</tr>";
 55     }
 56 }
 57 ?>
 58
 59 <!DOCTYPE html>
 60 <html lang="en">
 61
 62 <head>
 63     <meta charset="UTF-8">
 64     <meta name="viewport" content="width=device-width, initial-scale=1, shrink-to-fit=no">
 65     <!-- connect to stylesheets -->
 66     <link href='https://fonts.googleapis.com/css?family=League Spartan' rel='stylesheet'>
 67     <link rel="stylesheet" href="https://cdn.jsdelivr.net/npm/bootstrap-icons@1.11.2/font/bootstrap-icons.min.css">
 68     <link href="https://cdn.jsdelivr.net/npm/bootstrap@5.3.2/dist/css/bootstrap.min.css" rel="stylesheet" integrity="sha384-T3c6CoIi6uLrA9TheNEoa7RxnatzcDSCmG1MXxSR1GAsXEV/Dwykc2MPK8M2HN" crossorigin="anonymous">
 69     <link href='styles.css' rel='stylesheet'>
 70     <link rel="icon" href="logo/favicon3.png">
 71     <title>Dashboard - SoundBro</title>
 72 </head>
 73
 74 <body class="bg-light">
 75     <!-- display navigation bar -->
 76     <?php include "includes/navbar.php" ?>
 77
 78     <!-- Display welcome message -->
 79     <h1 class = "p-2 mt-4 text-center">Dashboard</h1>
 80     <h3 class = "mb-4 text-center text-secondary">Welcome, <?php echo $firstName ; ?>! </h3>
 81     <hr class="border-dark border-2">
 82
 83     <!-- display additional message for Ms Pearcey (admin) -->
 84     <?php if ($_SESSION['user_type']=='admin') { ?>
 85         <div class="container p-3 my-5 border">
 86             <p class="fs-5">
 87                 Welcome! If you are here for admin purposes, click <a href="dashboard-admin.php">here.
 88             </a> <br>
 89                 If not, stay on this page to see music teacher / accompanist info.
 90             </p>
 91             <hr>
 92         <?php }?>
 93
 94     <?php
 95
 96     // get basic info + start date
 97     $publishSQL="SELECT IsPublished,StartDate FROM BasicInfo";
 98     $publishResult=$conn->query($publishSQL);
 99     if ($publishResult and $publishResult->num_rows > 0){
100         $publishInfo=$publishResult->fetch_assoc();
101         $IsPublished=$publishInfo['IsPublished'];
102         $startDate=$publishInfo['StartDate'];
103         $timestamp = strtotime($startDate);
104         $formattedDate = date('jS M Y', $timestamp);
105     } else {
106         // schedule not created yet, so set the variable $IsPublished to 0
107         $IsPublished = 0;
108     }
```



```
109
110 // if user is a student
111 if ($_SESSION['user_type']=='student'){
112     $results_message="If you have recently taken an exam, results will be displayed on the
113     music notice board, or you will be contacted by your music teacher.";
114     //is schedule published yet
115     if ($IsPublished==1){ ?>
116         <!-- display STUDENT exam schedule -->
117         <div class="container mt-5">
118             <h2>Exam Schedule | <?php echo $formattedDate ; ?></h2>
119             <table class="table table-bordered">
120                 <thead>
121                     <!-- table headings -->
122                     <tr class='table-primary'>
123                         <th>Student Name</th>
124                         <th>Instrument</th>
125                         <th>Grade</th>
126                         <th>Accompanist Name</th>
127                         <th>Start Time</th>
128                     </tr>
129                 </thead>
130                 <tbody>
131                     <?php
132                         // get info from DB to create correct schedule format
133                         $examineeSQL="SELECT students.FirstName AS student_fname,students.LastName AS s
tudent_lname,Instrument,Grade,StartTime,examinees.AccompID
134                         FROM schedule
135                         JOIN examinees ON schedule.ExamineeID=examinees.ExamineeID
136                         JOIN students ON students.StudentID=examinees.ExamineeID
137                         WHERE examinees.ExamineeID='$ID';";
138                         $examineeResult = $conn->query($examineeSQL);
139                         // if result is not empty
140                         if ($examineeResult and $examineeResult->num_rows > 0) {
141                             // display schedule
142                             displaySchedule($conn,$examineeResult,false);
143                         } else {
144                             // no records found, display message
145                             echo "<tr><td colspan='5'>
146                                 You aren't scheduled to play in the upcoming exams.<br>$results_message</td
147                             </tr>";
148                         }
149                     ?>
150                     </tbody>
151                 </table>
152             </div><?php
153         } else {
154             // not published yet
155             $message.="Nothing has been scheduled yet. Check again later.<br>$results_message";
156         }
157 }
```



```
157 // user must be a supervisor / admin
158 } else {
159     // is schedule published?
160     if ($IsPublished==1){ ?>
161
162         <!-- display TEACHER exam schedule -->
163         <div class="container my-5">
164             <h2>Exam Schedule | <?php echo $formattedDate ; ?> | Teacher Info</h2>
165             <table class="table table-bordered">
166                 <thead>
167                     <!-- table headings -->
168                     <tr class='table-primary'>
169                         <th>Student Name</th>
170                         <th>Instrument</th>
171                         <th>Grade</th>
172                         <th>Teacher Name</th>
173                         <th>Accompanist Name</th>
174                         <th>Start Time</th>
175                     </tr>
176                 </thead>
177                 <tbody>
178                     <?php
179                         // get info from DB to create correct schedule format
180                         $teacherSQL="SELECT students.FirstName AS student_fname,students.LastName AS st
udent_lname,Instrument,Grade,StartTime,examinees.AccompID,ScheduleID,supervisors.FirstName AS t
eacher_fname,supervisors.LastName AS teacher_lname
181                         FROM schedule
182                         JOIN examinees ON schedule.ExamineeID=examinees.ExamineeID
183                         JOIN supervisors ON supervisors.SupervisorID=examinees.SupervisorID
184                         JOIN students ON students.StudentID=examinees.ExamineeID
185                         WHERE supervisors.SupervisorID='$ID'
186                         ORDER BY StartTime ASC";
187                         $teacherResult = $conn->query($teacherSQL);
188                         // if result is not empty
189                         if ($teacherResult and $teacherResult->num_rows > 0) {
190                             // display schedule
191                             displaySchedule($conn,$teacherResult,true);
192                         } else {
193                             // no records found, display message
194                             echo "<tr><td colspan='6'>
195                                 You aren't teaching any upcoming examinees.</td></tr>";
196                         }
197                     ?>
198                 </tbody>
199             </table>
200             </div>
201
202             <!-- display ACCOMPANIST exam schedule -->
203             <div class="container my-5">
204                 <h2>Exam Schedule | <?php echo $formattedDate ; ?> | Accompanist Info</h2>
205                 <table class="table table-bordered">
```



```
206         <thead>
207             <tr class='table-primary'>
208                 <!-- table headings -->
209                 <th>Student Name</th>
210                 <th>Instrument</th>
211                 <th>Grade</th>
212                 <th>Accompanist Name</th>
213                 <th>Start Time</th>
214             </tr>
215         </thead>
216         <tbody>
217             <?php
218                 // get info from DB to create correct schedule format
219                 $accompSQL="SELECT students.FirstName AS student_fname,students.LastName
220                     AS student_lname,Instrument,Grade,StartTime,examinees.AccompID,examinees.Examin
eeID,ScheduleID
221                     FROM schedule
222                     JOIN examinees ON schedule.ExamineeID=examinees.ExamineeID
223                     JOIN examaccompson ON examinees.AccompID=examaccompson.AccompID
224                     JOIN supervisors ON examaccompson.AccompID=supervisors.SupervisorID
225                     JOIN students ON students.StudentID=examinees.ExamineeID
226                     WHERE supervisors.SupervisorID='$ID'
227                     ORDER BY StartTime ASC";
228                     $accompResult = $conn->query($accompSQL);
229                     // if result is not empty
230                     if ($accompResult and $accompResult->num_rows > 0) {
231                         // display schedule
232                         displaySchedule($conn,$accompResult,false);
233                     } else {
234                         // no records found, display message
235                         echo "<tr><td colspan='5'>
236                             You aren't accompanying any upcoming exams.</td></tr>";
237                     }
238                     ?>
239             </tbody>
240         </table>
241         </div><?php
242     } else {
243         // nothing published yet
244         $message.= "Nothing has been scheduled yet. Check again later.";
245     }
246 }
247 ?>
248 <!-- display messages -->
249 <div class="container p-3 ">
250     <p class="text-center">
251         <?php echo $message."<br>";
```



```
254     // add a horizontal line to break up the text
255     if ($message != ""){ ?>
256         <hr>
257     <?php } ?>
258
259     <p class="fs-5 text-center">
260         Email Ms. Pearcey at <strong>admin@email.com</strong> if you have any concerns.
261     </p>
262     </p>
263     </div>
264     <!-- display footer -->
265     <?php include 'includes/footer.html'?>
266     </body>
267
268 </html>
```

## DASHBOARD-ADMIN.PHP

```
● ● ●
1 <?php
2 session_start();
3 // if logged out redirect to login.php
4 if (!isset($_SESSION['user_id'])) {
5     header("Location: login.php"); // Redirect to login page
6     exit();
7 }
8
9 // if not an admin, redirect to dashboard
10 if ($_SESSION['user_type']!='admin') {
11     //redirect to admin page
12     header("Location: dashboard.php");
13     exit();
14 }
15
16 include 'includes/db-connect.php';
17
18 //get first name for welcome message
19 $ID=$_SESSION['user_id'];
20 $result=$conn->query("SELECT * FROM Supervisors WHERE SupervisorID = '$ID'");
21 $row = $result->fetch_assoc();
22 $firstName=$row['FirstName'];
23
24 $scheduleExists=False;
25 // check status of scheduling process
26 $sql="SELECT * FROM BasicInfo";
27 $scheduleResult=$conn->query($sql);
28
29 // has schedule been made?
30 if ($scheduleResult -> num_rows > 0){
31     $scheduleExists=True;
32 }
33
34 // view existing schedule button
35 if(isset($_POST['viewExisting'])) {
36     // redirect to schedule-complete.php
37     header("Location: schedule-complete.php");
38     exit();
39 }
40
41 // start new schedule button
42 if (isset($_POST['new'])) {
43     //empty basic info and schedule
44     $clearBasicInfo="TRUNCATE TABLE BasicInfo";
45     $conn->query($clearBasicInfo);
46     $clearSchedule="TRUNCATE TABLE Schedule";
47     $conn->query($clearSchedule);
48     // redirect to schedule page
49     header("Location: schedule.php");
50     exit();
51 }
52 ?
53 ?>
```



```
54
55 <!DOCTYPE html>
56 <html lang="en">
57
58 <head>
59   <meta charset="UTF-8">
60   <meta name="viewport" content="width=device-width, initial-scale=1, shrink-to-fit=no">
61   <!-- connect to stylesheets -->
62   <link href='https://fonts.googleapis.com/css?family=League Spartan' rel='stylesheet'>
63   <link rel="stylesheet" href="https://cdn.jsdelivr.net/npm/bootstrap-icons@1.11.2/font/bootstrap-icons.min.css">
64   <link href="https://cdn.jsdelivr.net/npm/bootstrap@5.3.2/dist/css/bootstrap.min.css" rel="stylesheet" integrity="sha384-T3c6CoIi6uLrA9TheNEoa7RxnatzcDSCmG1MXxSR1GAsXEV/Dwykc2MPK8M2HN" crossorigin="anonymous">
65   <link href='styles.css' rel='stylesheet'>
66   <link rel="icon" href="logo/favicon3.png">
67   <title>Admin Dashboard - SoundBro</title>
68 </head>
69
70 <body class="bg-light">
71 <!-- display navigation bar -->
72 <?php include "includes/navbar.php" ?>
73
74 <!-- display if user has just published -->
75 <?php if (isset($_GET['published']) and $_GET['published']==1) { ?>
76   <div class="alert alert-success m-2 ">
77     The schedule has been published!
78   </div>
79 <?php } ?>
80
81 <!-- welcome message -->
82 <div class='p-2 m-5 text-center'>
83   <h2>Welcome, <?php echo $firstName ; ?>! </h2>
84   <!-- link back to dashboard -->
85   <p> This is the admin page. Click <a href="dashboard.php">here</a> to go to the music teacher / accompanist login. </p>
86 </div>
87
88 <div class="container my-5">
89   <div class="row justify-content-center">
90     <div class="col-md-7 bg-white p-5 rounded-4 shadow">
91       <h2 class="text-center mb-4 pb-4 text-primary border-3 border-bottom">Schedule Music Exams</h2>
92       <form method="post" action="" class='text-center'>
93
94         <!-- view existing schedule -->
95         <?php if ($scheduleExists) { ?>
96           <div class="m-2 p-2 form-group text-center">
97             <button type="submit" name="viewExisting" class="fs-5 btn p-2 btn-primary w-75 btn-block"><i class="bi bi-eye"></i> View Existing Schedule</button>
98           </div>
99         <?php } ?>
100
```

```
101      <!-- start new schedule -->
102      <div class="m-2 p-2 form-group text-center">
103          <button type="submit" name="new" class="fs-5 btn p-2 btn-danger w-75 btn-b
lock"><i class="bi bi-pencil-square"></i> Start New Schedule</button>
104      </div>
105      <?php if($scheduleExists){ ?>
106          <!-- warning message -->
107          <p><strong>Warning:</strong> Starting a new schedule will delete the existi
ng one.</p>
108      <?php } ?>
109      </form>
110  </div>
111 </div>
112 </div>
113
114
115 <!-- display footer -->
116 <?php include 'includes/footer.html'?>
117 </body>
118 </html>
```

## SCHEDULE.PHP



```
1 <?php
2 // Start the session to store user data
3 session_start();
4
5 // if logged out redirect to login.php
6 if (!isset($_SESSION['user_id'])) {
7     header("Location: login.php");
8     exit();
9 }
10 // if not an admin, redirect to dashboard
11 if ($_SESSION['user_type']!='admin') {
12     header("Location: dashboard.php");
13     exit();
14 }
15
16 // connect to database
17 include 'includes/db-connect.php';
18
19 // return ideal time of next break
20 function nextBreak($numCompletedBreaks){
21     $error = "";
22     // validate input
23     if ($numCompletedBreaks >= 0 and $numCompletedBreaks <= 3) {
24         // set next break time based on number of breaks completed
25         switch($numCompletedBreaks){
26             case 0:
27                 $breaktime = new DateTime('10:45');
28                 break;
29             case 1:
30                 $breaktime = new DateTime('12:30');
31                 break;
32             case 2:
33                 $breaktime = new DateTime('15:15');
34                 break;
35             case 3:
36                 $breaktime = new DateTime('17:00');
37                 break;
38         }
39     } else {
40         // create error message
41         $error = "Invalid number of completed breaks.<br> ";
42         // set breaktime to something to prevent errors
43         $breaktime = new DateTime('23:00');
44     }
45     return [$breaktime, $error];
46 }
47
```

```

48 // decide if schedule is meeting TCL regulations or not
49 function meetingRegulations($time,$dayExamTime,$sessionTime,$numCompletedBreaks){
50     $latestTime = new DateTime('17:00');
51     $latestLunchTime = new DateTime('12:45');
52
53     // decide if time is valid
54     $timeValid = $time->format('H:i') <= $latestTime->format('H:i');
55     // decide if a lunch break must be scheduled or not
56     $lunchValid = ($numCompletedBreaks != 1) || ($time->format('H:i') <= $latestLunchTime->format('H:i'));
57     // decide if the current session is too long
58     $sessionValid = $sessionTime <= 120;
59     // decide if too many exams have been scheduled
60     $dayExamValid = $dayExamTime <= 390;
61
62     return $timeValid and $lunchValid and $sessionValid and $dayExamValid ;
63 }
64
65 // schedule breaks
66 function addBreak($time,$numCompletedBreaks){
67     $eod=False;
68     $sessionTime=0; // scheduling a break, so current session ends
69     $error="";
70     // validate input
71     if ($numCompletedBreaks >= 0 and $numCompletedBreaks <= 3) {
72         switch($numCompletedBreaks){
73             case 0:
74                 // 15 min break
75                 $time->modify('+15 minutes');
76                 $numCompletedBreaks+=1;
77                 break;
78             case 1:
79                 // 60 min lunch break
80                 $time->modify('+60 minutes');
81                 $numCompletedBreaks+=1;
82                 break;
83             case 2:
84                 // 15 min break
85                 $time->modify('+15 minutes');
86                 $numCompletedBreaks+=1;
87                 break;
88             case 3:
89                 // end of day
90                 $eod=True;
91                 break;
92         }
93     } else {
94         // invalid numCompletedBreaks
95         $error = "Invalid number of completed breaks. <br>";
96     }
97     return [$time,$sessionTime,$numCompletedBreaks,$eod,$error];
98 }
99

```

```

100 // get field in ExamAccomps corresponding to the time
101 function getAccompField($time){
102
103     // round down minutes to nearest 30
104     $minute = $time->format('i');
105     $roundedMinute = floor($minute / 30) * 30;
106
107     // create new object with rounded minutes
108     $roundDownTime = clone $time;
109     $roundDownTime->setTime(
110         $time->format('H'),
111         $roundedMinute,
112         $time->format('s')
113     );
114
115     // format time as 'StartHHMM'
116     $formattedTime = 'Start' . $roundDownTime->format('Hi');
117     return [$roundDownTime,$formattedTime];
118 }
119
120 // get data about an examinee from DB
121 function getExamineeInfo($ExamineeID,$conn){
122
123     // get Duration + AccompID of examinees exam
124     $examineeinfo="SELECT Duration,Examinees.AccompID
125     FROM timings
126     JOIN families ON families.InstrumentFamily = timings.InstrumentFamily
127     JOIN examinees ON examinees.Instrument = families.Instrument AND timings.Grade=examinees.G
128     WHERE ExamineeID='$ExamineeID'";
129     $examineeResult=$conn->query($examineeinfo);
130     $row=$examineeResult->fetch_assoc();
131     // retrieve variables from SQL result
132     $duration=$row['Duration'];
133     $accompID=$row['AccompID'];
134
135     return [$duration,$accompID];
136 }
137
138 // determine if an examinee's accompanist is available at a given time
139 function accompAvailable($ExamineeID,$time,$conn){
140
141     // get info about examinee
142     [$duration,$accompID]=getExamineeInfo($ExamineeID,$conn);
143
144     // create object for midpoint time
145     $midTime = clone $time;
146     $halfDuration=$duration / 2;
147     $midTime->modify("+$halfDuration} minutes");
148
149

```

```

149     // create object for end time
150     $endTime = clone $time;
151     $endTime->modify("+$duration minutes");
152
153     //get field name for start/mid/end time
154     [$roundDownTime,$startField]=getAccompField($time); // $time is the start time of the exam
155     [$roundDownTime,$midField]=getAccompField($midTime);
156     [$roundDownTime,$endField]=getAccompField($endTime);
157     //roundDownTime is ignored (it is returned to simplify the function recalculateOrder)
158
159     // if exams run past 5pm
160     if (($startField=='Start1700' || $midField=='Start1700' || $endField=='Start1700')
161         and ($endTime->format('H:i')!='17:00')) {
162         // this breaks regulations so return false
163         return false;
164     } else {
165         if (!$accompID){
166             // examinee has no accompanist, return true
167             return true;
168         } else {
169             if ($endTime->format('H:i')=='17:00'){
170                 // if exam ends at 5pm, remove 'Start1700'
171                 $endField=$midField;
172             }
173             // get availability from ExamAccomps
174             $availabilitySQL="SELECT $startField,$midField,$endField
175             FROM ExamAccomps
176             WHERE AccompID = $accompID";
177             $availabilityResult=$conn->query($availabilitySQL);
178             $availabilityRow=$availabilityResult->fetch_assoc();
179
180             // store 3 variables to indicate availability at each point
181             $available_start = $availabilityRow[$startField];
182             $available_mid = $availabilityRow[$midField];
183             $available_end = $availabilityRow[$endField];
184
185             if ($available_start==1 and $available_mid==1 and $available_end==1){
186                 // if accompanist is available the during the whole exam
187                 return true;
188             } else {
189                 // if accompanist is not available during exam at any point
190                 return false;
191             }
192         }
193     }
194 }
195

```



```
196 // sort accompanists appropriately
197 function freeTimeSort($rowA, $rowB) {
198     $leniency_threshold=150;
199
200     if ($rowA[3] > $leniency_threshold and $rowB[3] > $leniency_threshold){
201         // free time is greater than threshold, return 0 to keep order unchanged
202         return 0;
203     } else {
204         // free time is below threshold, sort by free time to maximise efficiency
205         return $rowA[3] - $rowB[3];
206     }
207 }
208
209 // sort all accompanists by free time at the start
210 function freeTimeSort_initial($rowA,$rowB){
211     // sort ascending by free time
212     return $rowA[3] - $rowB[3];
213 }
214
215 // sort examinees by family, instrument and then grade
216 function instrumentSort($a, $b) {
217     // sort by instrument family
218     $result = strcmp($a['InstrumentFamily'], $b['InstrumentFamily']);
219
220     // if family is the same, sort by instrument
221     if ($result === 0) {
222         $result = strcmp($a['Instrument'], $b['Instrument']);
223     }
224
225     // if instrument is the same, sort ascending by grade
226     if ($result === 0) {
227         $result = $a['Grade'] - $b['Grade'];
228     }
229
230     return $result;
231 }
232
```



```
233 // determine order of priority for scheduling examinees
234 function recalculateOrder($examineeIDs,$time,$conn){
235     $accompIDs=[];
236     $error="";
237
238     //calculate total exam time
239     for ($i = 0; $i < count($examineeIDs); $i++) {
240         $ExamineeID=$examineeIDs[$i];
241         // get AccompID and Duration for examinee
242         [$duration,$accompID]=getExamineeInfo($ExamineeID,$conn);
243
244         // if accompID null, do not add to accompIDs
245         if ($accompID){
246
247             $accompIndex = null;
248             // iterate through each accompanist
249             foreach ($accompIDs as $rIndex => $row) {
250                 // Check if this row matches the accompID
251                 if ($row[0] === $accompID) {
252                     // increment by duration
253                     $accompIndex = $rIndex;
254                     $accompIDs[$accompIndex][1] += $duration;
255                     break;
256                 }
257             }
258             // If accompID not yet in accompIDs
259             if ($accompIndex === null) {
260                 // add to array, and increment by duration
261                 $accompIDs[]=$accompID,0,0,0];
262                 $accompIndex=count($accompIDs)-1;
263                 $accompIDs[$accompIndex][1] += $duration;
264             }
265         }
266     }
267
268     // calculate time until next 30min slot
269     [$difference,$currentField]=timeDifference($time);
270
271     if ($time->format('H:i')!="17:00"){
272         //calculate available time
273         for ($i = 0; $i < count($accompIDs); $i++){
274
275             // get availability for accompanist
276             $accompID=$accompIDs[$i][0];
277             $availabilitySQL="SELECT Start0900,Start0930,Start1000,Start1030,Start1100,
278             Start1130,Start1200,Start1230,Start1300,Start1330,Start1400,
279             Start1430,Start1500,Start1530,Start1600,Start1630
280             FROM ExamAccomps
281             WHERE AccompID=$accompID";
282
283             $availabilityResult=$conn->query($availabilitySQL);
284
285             // get associative array of availability
286             $availability=$availabilityResult->fetch_assoc();
287
288             // if accomp is currently available, add $difference to their available time
289             if ($availability[$currentField]==1){
290                 $accompIDs[$i][2]+=$difference;
291             }
292     }
```



```
293     // shorten array to everything after currentField
294     $currentIndex = array_search($currentField, array_keys($availability));
295     $availability=array_slice($availability,$currentIndex+1);
296
297     foreach ($availability as $key => $value) {
298         //increment available time by 30 for every available slot
299         if ($availability[$key]==1){
300             $accompIDs[$i][2]+=30;
301         }
302     }
303 }
304 }
305
306 //calculate free time
307 for ($i = 0; $i < count($accompIDs); $i++){
308     $accompID=$accompIDs[$i][0];
309     $examTime = $accompIDs[$i][1];
310     $availableTime = $accompIDs[$i][2];
311     $freeTime = $availableTime - $examTime;
312
313     if ($freeTime < 0){
314         $minutes_more=$freeTime*-1;
315
316         //get name from DB
317         $nameSQL="SELECT FirstName, LastName
318         FROM Supervisors
319         WHERE SupervisorID = $accompID";
320         $nameResult=$conn->query($nameSQL);
321         $nameRow=$nameResult->fetch_assoc();
322         $firstName=$nameRow['FirstName'];
323         $lastName=$nameRow['LastName'];
324
325         //update error message with relevant info
326         $error.="$firstName $lastName has too many exams / is not available enough.
327         They need <u>at least</u> $minutes_more minutes more. <br>";
328     }
329     // update array to include freeTime
330     $accompIDs[$i][3]=$freeTime;
331 }
332
333
334 // sort ascending by free time
335 if ($time->format('H:i')=='09:00'){
336     //sort strictly at the start
337     usort($accompIDs,'freeTimeSort_initial');
338 } else {
339     //sort with some leniency
340     usort($accompIDs,'freeTimeSort');
341 }
342
343 $newExamineeIDs=[];
344 //convert sorted accompIDs into sorted examineeIDs
345 for ($i = 0; $i < count($accompIDs); $i++){
346     $temp_array=[];
347 }
```

```
348     for ($j=0; $j < count($examineeIDs); $j++){
349
350         //find examinees in examineeIDs with current accompanist
351         $accompSQL="SELECT AccompID FROM Examinees
352         WHERE ExamineeID='".$examineeIDs[$j]."'";
353         $accompResult=$conn->query($accompSQL);
354         $accompRow=$accompResult->fetch_assoc();
355         $examinee_accomp=$accompRow['AccompID'];
356
357         if ($examinee_accomp==$accompIDs[$i][0]){
358             //add their info to temp_array
359             $instrumentSQL="SELECT Examinees.ExamineeID,
360             Families.InstrumentFamily,Families.Instrument,Timings.Grade
361             FROM timings
362             JOIN families ON families.InstrumentFamily = timings.InstrumentFamily
363             JOIN examinees ON examinees.Instrument=families.Instrument
364             AND timings.Grade=examinees.Grade
365             WHERE examinees.ExamineeID='".$examineeIDs[$j]."'";
366             $instrumentResult=$conn->query($instrumentSQL);
367             $temp_array[]=$instrumentResult->fetch_assoc();
368         }
369     }
370
371     //sort by family > instrument > grade
372     usort($temp_array, 'instrumentSort');
373
374     //temp_array is added to the end of newExamineeIDs
375     $newExamineeIDs = array_merge($newExamineeIDs, $temp_array);
376     //re-index the array so it is formatted correctly
377     $newExamineeIDs = array_values($newExamineeIDs);
378 }
379
380 //repeat for examinees with no accompanist
381 $temp_array=[];
382 for ($i=0; $i < count($examineeIDs); $i++){
383
384     //find examinees in examineeIDs with no accompanist
385     $accompSQL="SELECT AccompID FROM Examinees
386     WHERE ExamineeID='".$examineeIDs[$i]."'";
387     $accompResult=$conn->query($accompSQL);
388     $accompRow=$accompResult->fetch_assoc();
389     $examinee_accomp=$accompRow['AccompID'];
390
391     if ($examinee_accomp==null){
392         //add their info to temp_array
393         $instrumentSQL="SELECT Examinees.ExamineeID,Families.InstrumentFamily,
394             Families.Instrument,Timings.Grade
395             FROM timings
396             JOIN families ON families.InstrumentFamily = timings.InstrumentFamily
397             JOIN examinees ON examinees.Instrument = families.Instrument
398             AND timings.Grade=examinees.Grade
399             WHERE examinees.ExamineeID='".$examineeIDs[$i]."'";
400             $instrumentResult=$conn->query($instrumentSQL);
401             $temp_array[]=$instrumentResult->fetch_assoc();
402     }
403 }
```

```

405     //sort by family > instrument > grade
406     usort($temp_array, 'instrumentSort');
407
408     //temp_array is added to the end of newExamineeIDs
409     $newExamineeIDs = array_merge($newExamineeIDs, $temp_array);
410     //re-index the array so it is formatted correctly
411     $newExamineeIDs = array_values($newExamineeIDs);
412
413     //extract just the ExamineeIDs
414     $finalExamineeIDs = [];
415     foreach ($newExamineeIDs as $subArray) {
416         $finalExamineeIDs[] = $subArray['ExamineeID'];
417     }
418     //var_dump($accompIDs);
419     return [$finalExamineeIDs,$error];
420 }
421
422 // find number of minutes until next 30min slot
423 function timeDifference($time){
424     // get rounded time, and field corresponding to time
425     [$roundDownTime,$currentField]=getAccompField($time);
426     //adjust roundDownTime so it stores time of next field
427     $roundDownTime->modify('+30 minutes');
428
429     //find difference between current time and next 30min slot
430     $difference = $time->diff($roundDownTime);
431     $difference=$difference->i; //format so it stores number of minutes
432     return [$difference,$currentField];
433 }
434
435 // Schedule exams
436 function schedule($conn){
437
438     // empty Schedule table in DB
439     $emptySchedule="TRUNCATE TABLE Schedule";
440     $conn->query($emptySchedule);
441
442     // initialise variables/classes/constants
443     $numCompletedBreaks=0;
444     $time = new DateTime('09:00');
445     $sessionTime=0;
446     $dayExamTime=0;
447     $eod=false;
448     $error="";
449     $interval = new DateInterval("PT10M");
450
451     //create examineeIDs
452     $examineeSQL="SELECT ExamineeID FROM Examinees";
453     $examineeResult=$conn->query($examineeSQL);
454     $examineeIDs = [];
455     while ($row = $examineeResult->fetch_assoc()) {
456         $examineeIDs[] = $row['ExamineeID'];
457     }
458

```



```
459 // loop until everyone is scheduled / end of day / error is found
460 while (count($examineeIDs)!=0 and $eod==false and $error==""){
461
462     //sort examinees by priority and other factors
463     [$examineeIDs,$miniError]=recalculateOrder($examineeIDs,$time,$conn);
464     $error.= $miniError; //append miniError to error
465
466     //search for available accompanist
467     $available=false;
468     $i=0;
469     // loop until available accompanist is found / end of list
470     while ($available == false and $i<count($examineeIDs)){
471         $available = accompAvailable($examineeIDs[$i],$time,$conn);
472         if (!$available){
473             //iterate through examineeIDs
474             $i++;
475         }
476     }
477
478     // if available accompanist is found
479     if($available){
480         $ExamineeID=$examineeIDs[$i];
481
482         //get duration of examinee
483         [$duration,$accompID] = getExamineeInfo($ExamineeID,$conn);
484
485         //validate the possible schedule change
486         $newTime= clone $time;
487         $newTime->modify("+{$duration} minutes");
488         $newSessionTime = $sessionTime + $duration;
489         $newDayExamTime = $dayExamTime + $duration;
490         $isValid = meetingRegulations($newTime,$newDayExamTime,
491         $newSessionTime,$numCompletedBreaks);
492
493         // get ideal time of next break
494         [$nextBreak,$miniError]=nextBreak($numCompletedBreaks);
495         $error.= $miniError;
496         // get latest time to schedule a break
497         $upperBound = $nextBreak->add($interval);
498
499         // if new schedule is valid and it is not time for a break
500         if ($newTime<=$upperBound and $isValid){
501             // Schedule an exam
502
503             // insert into schedule
504             $StartTime=$time->format('H:i');
505             $schedulingSQL="INSERT INTO Schedule (StartTime,ExamineeID)
506             VALUES ('$StartTime','$ExamineeID')";
507             $conn->query($schedulingSQL);
508
509             // remove examinee from examineeIDs
510             unset($examineeIDs[$i]);
511             $examineeIDs=array_values($examineeIDs);
512
513
```



```
513         // update time variables
514         $time=$newTime;
515         $sessionTime=$newSessionTime;
516         $dayExamTime=$newDayExamTime;
517
518     } else {
519         // Schedule a break
520         [$time,$sessionTime,$numCompletedBreaks,$eod,$miniError]
521         =addBreak($time,$numCompletedBreaks);
522         $error.= $miniError;
523     }
524 } else {
525     //find bounds for allowed break times
526     [$nextBreak,$miniError]=nextBreak($numCompletedBreaks);
527     $error.= $miniError;
528     $upperBound = $nextBreak->add($interval);
529     $lowerBound = $nextBreak->sub($interval);
530
531     //can a break be scheduled now?
532     if ($time >= $lowerBound and $time <= $upperBound){
533         [$time,$sessionTime,$numCompletedBreaks,$eod,$miniError]
534         =addBreak($time,$numCompletedBreaks);
535         $error.= $miniError;
536     } else {
537         //find the time difference until the next 30 min slot
538         [$difference,$currentField]=timeDifference($time);
539
540         //validate the possible schedule change
541         $newTime= clone $time;
542         $newTime->modify("+$difference minutes");
543         $newSessionTime = $sessionTime + $difference;
544         $isValid = meetingRegulations($newTime,$dayExamTime,
545         $newSessionTime,$numCompletedBreaks);
546
547         if ($isValid){
548             //if valid, increment time to next 30min slot
549             $time=$newTime;
550             $sessionTime=$newSessionTime;
551         } else {
552             //if invalid, schedule a break
553             [$time,$sessionTime,$numCompletedBreaks,$eod,$miniError]
554             =addBreak($time,$numCompletedBreaks);
555             $error.= $miniError;
556         }
557     }
558 }
559 }
560
561
```



```
561     $alert="";
562     if ($eod){
563         //display end of day message
564         $alert.= "The day has ended.<br> ";
565     }
566     if (count($examineeIDs) != 0){
567         // display unscheduled examinee names
568         $names=[];
569         for ($i=0 ; $i<count($examineeIDs) ; $i++) {
570             // get examinee name
571             $getNames = "SELECT FirstName, LastName
572             FROM Students WHERE StudentID='".$examineeIDs[$i]';
573             $nameResult = $conn->query($getNames);
574             $nameRow=$nameResult->fetch_assoc();
575             $fname=$nameRow['FirstName'];
576             $lname=$nameRow['LastName'];
577             //add name to array
578             $names[]="$fname $lname";
579         }
580         //add all names to alert message
581         $alert.= "The following examinees have not been scheduled: "
582             .implode(" , ",$names)."<br>";
583     } else {
584         // examineeIDs is empty, so everyone has been scheduled
585         $alert.= "All examinees have been scheduled!";
586     }
587
588     return [$alert,$error];
589 }
590 }
```



```
591 // schedule exam on button click
592 if ($_SERVER['REQUEST_METHOD'] === 'POST') {
593
594     // retrieve start date + prevent SQL injection
595     $startDate = mysqli_real_escape_string($conn,$_POST['basicinfo']);
596
597     $validationError="";
598     $today = new DateTime();
599     // presence check
600     if ($startDate==""){
601         $validationError="Please enter a date";
602     } else {
603         // reject if date is in the past
604         if ($startDate < $today->format('Y-m-d')){
605             $validationError="The date must be in the future.";
606         } else {
607             // check if examinees is empty
608             $examineeSQL="SELECT * FROM examinees";
609             $examineeResult=$conn->query($examineeSQL);
610             if ($examineeResult->num_rows == 0 ){
611                 $validationError="There is no examinee data in the database.";
612             } else {
613                 //enter data into BasicInfo
614                 $enterBasicinfo="INSERT INTO basicinfo (StartDate) VALUES('$startDate')";
615                 $conn->query($enterBasicinfo);
616                 // create schedule
617                 [$alert,$error]=schedule($conn);
618                 // redirect to schedule-complete.php with messages
619                 $_SESSION['alert']=$alert;
620                 $_SESSION['error']=$error;
621                 header("Location:schedule-complete.php");
622                 exit();
623             }
624         }
625     }
626 }
627
628 ?>
629
630 <!DOCTYPE html>
631 <html lang="en">
632
633 <head>
634     <meta charset="UTF-8">
635     <meta name="viewport" content="width=device-width, initial-scale=1, shrink-to-fit=no">
```



```
636     <!-- connect to stylesheets -->
637     <link href='https://fonts.googleapis.com/css?family=League Spartan' rel='stylesheet'>
638     <link rel="stylesheet" href="https://cdn.jsdelivr.net/npm/bootstrap-icons@1.11.2/font/bootstrap-icons.min.css">
639     <link href="https://cdn.jsdelivr.net/npm/bootstrap@5.3.2/dist/css/bootstrap.min.css" rel="stylesheet"
640           integrity="sha384-T3c6CoIi6uLrA9TheNEoa7RxnatjcdSCmG1MXxSR1GAsXEV/Dwykc2MPK8M2HN" crossorigin
641           origin="anonymous">
642     <link href='styles.css' rel='stylesheet'>
643     <link rel="icon" href="logo/favicon3.png">
644     <title>Schedule Exams-SoundBro</title>
645   </head>
646
647   <body class="bg-light">
648
649     <!-- link to navigation bar -->
650     <?php include 'includes/navbar.php' ?>
651
652     <div class="container my-5">
653       <div class="row p-5 justify-content-center">
654         <div class="col-md-8 bg-white p-5 rounded-4 shadow">
655           <h2 class="text-center mb-5 text-primary ">Schedule Music Exams</h2>
656           <form method="post" action="" class="text-center">
657
658             <!-- display if error is found -->
659             <?php if (isset($validationError) and $validationError!="") { ?>
660               <div class="alert alert-danger">
661                 <?php echo $validationError; ?>
662               </div>
663             <?php } ?>
664
665             <!-- enter start date -->
666             <div class="form-group p-3">
667               <label for="basicinfo"> Enter the start date</label>
668               <input type="date" class="form-control" id="basicinfo" name="basicinfo" >
669             </div>
670
671             <!-- message for entering data -->
672             <div class='fw-semibold fst-italic mt-4 p-3'>
673               Please enter examinee and accompanist data directly into the database!
674             </div>
675
676             <!-- schedule button -->
677             <div class="mt-5 form-group text-center">
678               <button type="submit" name="schedule" class="fs-4 w-75 btn btn-primary border border-4 rounded-pill">
679                 <i class="bi bi-magic"></i> Schedule</button>
680             </div>
681           </form>
682         </div>
683       </div>
684     </div>
685
686     <!-- display footer -->
687     <?php include 'includes/footer.html'?>
688   </body>
689 </html>
```

## SCHEDULE-COMPLETE.PHP

```
● ● ●  
1 <?php  
2 session_start();  
3 // if logged out redirect to login.php  
4 if (!isset($_SESSION['user_id'])) {  
5     header("Location: login.php");  
6     exit();  
7 }  
8 // if not an admin, redirect to dashboard  
9 if ($_SESSION['user_type']!='admin') {  
10    header("Location: dashboard.php");  
11    exit();  
12 }  
13  
14 // connect to database  
15 include 'includes/db-connect.php';  
16  
17 //get alert and error  
18 if (isset($_SESSION['alert']) and isset($_SESSION['error'])){  
19     $alert=$_SESSION['alert'];  
20     $error=$_SESSION['error'];  
21 } else {  
22     // confirm schedule has been made  
23     $sql="SELECT * FROM Schedule";  
24     $scheduleExists=$conn->query($sql);  
25     if ($scheduleExists -> num_rows == 0){  
26         // schedule not made, redirect to admin dashboard  
27         header("Location: dashboard-admin.php");  
28         exit();  
29     }  
30 }  
31  
32 // get start date of exam  
33 $basicinfoSQL="SELECT * FROM BasicInfo";  
34 $basicinfoResult=$conn->query($basicinfoSQL);  
35 $basicinfo=$basicinfoResult->fetch_assoc();  
36 $startDate=$basicinfo['StartDate'];  
37 // format start date appropriately  
38 $timestamp = strtotime($startDate);  
39 $formattedDate = date('jS M Y', $timestamp);  
40  
41 // is schedule published?  
42 $IsPublished=$basicinfo['IsPublished'];  
43
```

```
44 // schedule again button
45 if (isset($_POST['new'])) {
46     //empty basic info and schedule
47     $clearBasicInfo="TRUNCATE TABLE BasicInfo";
48     $conn->query($clearBasicInfo);
49     $clearSchedule="TRUNCATE TABLE Schedule";
50     $conn->query($clearSchedule);
51     // redirect to schedule page
52     header("Location: schedule.php");
53     exit();
54 }
55
56 //publish schedule button
57 if(isset($_POST['publish'])){
58     // set IsPublished to 1
59     $sql="UPDATE basicinfo SET IsPublished=1";
60     $conn->query($sql);
61     //redirect to admin dashboard
62     header("Location: dashboard-admin.php?published=1");
63     exit();
64 }
65
66 ?>
67
68
69 <!DOCTYPE html>
70 <html lang="en">
71
72 <head>
73     <meta charset="UTF-8">
74     <meta name="viewport" content="width=device-width, initial-scale=1, shrink-to-fit=no">
75     <!-- connect to stylesheets -->
76     <link href='https://fonts.googleapis.com/css?family=League Spartan' rel='stylesheet'>
77     <link rel="stylesheet" href="https://cdn.jsdelivr.net/npm/bootstrap-icons@1.11.2/font/bootstrap-icons.min.css">
78     <link href="https://cdn.jsdelivr.net/npm/bootstrap@5.3.2/dist/css/bootstrap.min.css" rel="stylesheet"
79         integrity="sha384-T3c6CoIi6uLrA9TneNEoa7RxnatzjcDSCmG1MXxSR1GAsXEV/Dwykc2MPK8M2HN" crossorigin="anonymous">
80     <link href='styles.css' rel='stylesheet'>
81     <link rel="icon" href="logo/favicon3.png">
82     <title>Schedule Complete -SoundBro</title>
83 </head>
84
85 <body class="bg-light">
86
87 <!-- link to navigation bar -->
88 <?php include 'includes/navbar.php' ?>
89 <h1 class="text-center m-4 text-primary">Scheduling Results</h1>
90
91 <!-- display if alert is found -->
92 <?php if (isset($alert) and $alert!="") { ?>
93     <div class="alert alert-info m-3">
94         <?php echo $alert; ?>
95     </div>
96 <?php } ?>
97
98
```



```
99  <!-- display if error is found -->
100 <?php if (isset($error) and $error!="") { ?>
101     <div class="alert alert-danger m-3">
102         <?php
103             echo "ERROR: <br>".$error;
104         ?>
105     </div>
106 <?php } ?>
107
108 <!-- display exam schedule -->
109 <div class="container mt-5">
110     <h2>Exam Schedule | <?php echo $formattedDate ; ?></h2>
111     <table class="table table-bordered">
112         <thead>
113             <!-- table headings -->
114             <tr class='table-primary'>
115                 <th>Student Name</th>
116                 <th>Instrument</th>
117                 <th>Grade</th>
118                 <th>Accompanist Name</th>
119                 <th>Start Time</th>
120             </tr>
121         </thead>
122         <tbody>
123             <?php
124                 // get info from DB to create correct schedule format
125                 $scheduleInfoSQL = "SELECT FirstName,LastName,examinees.Instrument,examinees.Grade,Star
tTime,Duration,AccompID
126                 FROM students
127                 JOIN examinees ON students.StudentID = examinees.ExamineeID
128                 JOIN schedule ON examinees.ExamineeID = schedule.ExamineeID
129                 JOIN families ON examinees.Instrument = families.Instrument
130                 JOIN timings ON timings.Grade=examinees.Grade AND families.InstrumentFamily=timings.In
strumentFamily;";
131                 $scheduleInfo = $conn->query($scheduleInfoSQL);
132
133                 if ($scheduleInfo->num_rows > 0) {
134
135                     $time= new DateTime('09:00');
136                     while ($row = $scheduleInfo->fetch_assoc()) {
137
138                         //get start time in correct form
139                         $StartTime=substr($row['StartTime'],0,5);
140                         $duration=$row['Duration'];
141
142                         // if times dont match, add an empty row
143                         if ($time->format('H:i') != $StartTime){
144                             // empty row to signal a break
145                             echo "<tr><td colspan='5'><br></td></tr>";
146                             // reset time
147                             $StartTime_object = DateTime::createFromFormat('H:i', $StartTime);
148                             $time->setTime($StartTime_object->format('H'), $StartTime_object->format
('i'));
149                         }
150                 }
```



```
151         // display info on table
152         echo "<tr>";
153         echo "<td>{$row['FirstName']} {$row['LastName']}</td>";
154         $instrument=ucwords($row['Instrument']);
155         echo "<td>{$instrument}</td>";
156         echo "<td>{$row['Grade']}</td>";
157
158         $AccompID=$row['AccompID'];
159         if ($AccompID){
160             // if examinee has accompanist, display their name
161             $accompInfoSQL="SELECT FirstName,LastName FROM Supervisors
162             JOIN ExamAccomps ON Supervisors.SupervisorID = ExamAccomps.AccompID
163             WHERE AccompID = '$AccompID'";
164             $accompInfo=$conn->query($accompInfoSQL);
165             $accompRow=$accompInfo->fetch_assoc();
166             echo "<td>{$accompRow['FirstName']} {$accompRow['LastName']}</td>";
167         } else {
168             // empty cell to maintain format
169             echo "<td> </td>";
170         }
171
172         //display start time
173         echo "<td><strong>{$StartTime}</strong></td>";
174         echo "</tr>";
175
176         //increment time
177         $time->modify("+$duration minutes");
178     }
179 } else {
180     // no records found so display message
181     echo "<tr><td colspan='5'>No records found</td></tr>";
182 }
183 ?>
184 </tbody>
185 </table>
186 </div>
187
188 <!-- display buttons -->
189 <form method="post" action="" class='text-center'>
190     <div class="d-grid " >
191         <div class="form-group text-center">
192             <!-- if nothing has been published -->
193             <?php if ($IsPublished==0) { ?>
194                 <!-- schedule again button -->
195                 <button type="submit" name="new" class="m-2 p-2 fs-4 w-25 btn btn-danger rounded-pill">
196                     <i class="bi bi-pencil-square"></i> Schedule Again
197                 </button>
198                 <!-- publish schedule button-->
199                 <button type="submit" name="publish" class="fs-4 m-2 p-2 btn w-25 btn-primary border rounded-pill">
200                     <i class="bi bi-send"></i> Publish Schedule
201                 </button>
202             <?php } ?>
203         </div>
204     </div>
205 </form>
206
```



```
206
207 <!-- display footer -->
208 <?php include 'includes/footer.html'?>
209 </body>
210 </html>
```

## LOGOUT.PHP



```
1 <?php
2 session_start();
3 session_unset(); // unset all session variables
4 session_destroy(); // destroy the session
5 header("Location: index.php"); // Redirect to index page
6 exit();
7 ?>
```

## STYLES.CSS



```
1 /* format image to be displayed correctly */
2 .biglogo{
3     display:block;
4     max-width:60%;
5     margin: 20px auto;
6     padding:20px;
7 }
8
9 /*Same font as SoundBro logo for consistency*/
10 .logo{
11     font-family:'League Spartan';
12     letter-spacing: -1px;
13 }
14
15 /* adding underline on hover makes it easier to see*/
16 .nav-item:hover{
17     text-decoration: underline;
18     text-decoration-color: white;
19 }
20
21
```

## DB-CONNECT.PHP



```
1 <?php
2 // connect to database using credentials
3 $serverName = "localhost";
4 $serverUsername = "root";
5 $serverPassword = "";
6 $dbname = "db_winter_23";
7 // set up connection to database
8 $conn = new mysqli($serverName, $serverUsername, $serverPassword,
9 $dbname);
10 // unexpected fatal error
11 if ($conn->connect_error) {
12     // display error message
13     die("Connection failed: " . $conn->connect_error);
14 }
15 ?>
```

## FOOTER.HTML



```
1 <!-- footer -->
2 <footer class="text-center" style="background-color: rgba(0,0,0,0.05);">
3     <i class="bi bi-music-note-beamed fs-2 mb-2"></i>
4     <div class="fs-5">
5         Rohan Desai 2023
6     </div>
7     <p class="fs-6 text-muted fst-italic">
8         King Edward VI Camp Hill School for Boys
9     </p>
10 </footer>
```

## NAVBAR.PHP



```
1  <!-- navigation bar -->
2  <nav class="navbar navbar-expand-sm bg-primary navbar-dark">
3      <!-- display links for navbar -->
4      <ul class="navbar-nav">
5          <li class="nav-item">
6              <!-- display SoundBro -->
7              <a class="nav-link logo fw-bolder fs-4 navbar-brand ms-2"" href="index.php">
8                  SoundBro.
9              </a>
10         </li>
11         <li class="nav-item">
12             <!-- display link to dashboard -->
13             <a class="nav-link me-2" href="dashboard.php">
14                 <i class="bi bi-music-note-list"></i> Dashboard
15             </a>
16         </li>
17         <?php
18             // Check if the user is logged in
19             if (isset($_SESSION['user_id'])) {
20                 // logged in, so display 'logout'
21                 echo '<li class="nav-item"><a href="logout.php" class="nav-link me-2"><i class="bi bi-box-arrow-right"></i> Logout</a></li>';
22             } else {
23                 // logged out, so display 'login' and 'register'
24                 echo '<li class="nav-item"><a class="nav-link me-2" href="login.php"><i class="bi bi-box-arrow-in-right"></i> Login</a></li><li class="nav-item"><a class="nav-link me-2" href="register.php"><i class="bi bi-pen"></i> Register</a></li>';
25             }
26             ?>
27         </ul>
28     </nav>
29
30
```