

APPOINTMENT SCHEDULER AND MANAGER – AN ANDROID APPLICATION

GRADUATE PROJECT

Submitted to the Faculty of
the Department of Computing Sciences
Texas A&M University-Corpus Christi
Corpus Christi, Texas

In Partial Fulfillment of the Requirements for the Degree of
Master of Science in Computer Science

By

Sahana Alikanti
Spring 2017

Committee Members

Dr. Dulal C Kar
Committee Chairperson

Dr. David Thomas
Committee Member

ABSTRACT

As a professor/employer in an institution, one would be busy most of the time with their personal works and responsibilities. Scheduling appointments for the people who would like to meet them seeking professional advice or help should not be an extra burden for them. Analyzing their schedule and providing appointments would be a time-consuming task. The main idea of this android application is to save them some time by scheduling the appointments automatically. This can be achieved by checking the Professor's free/office hours when somebody requests an appointment. When a person requests an appointment, the employer (whom the person requested) will be notified about the request and once they approve, it will be automatically scheduled. This application can also check for appropriate time slots as requested by the user in special cases and approves his request.

The main idea of this application is to save one's time where he can request, postpone or cancel an appointment by just a single click. If there is more than 5-minute delay (as we do not want to disturb all others just because of 2-3 min delay) in one appointment, all the users down the list whose appointments would be affected by this delay will be notified about the delay. This application also deals with priorities, for instance, if an appointment gets cancelled due to the prior appointment's time-delay, the one will be given higher priority on the next day. It also manages to avoid time conflicts if two users are requesting appointment for the same time.

TABLE OF CONTENTS

ABSTRACT.....	i
TABLE OF CONTENTS.....	ii
LIST OF FIGURES	v
LIST OF TABLES.....	vii
1. BACKGROUND AND RATIONALE.....	1
1.1. Existing Applications.....	1
1.1.1. SetMore Appointments	1
1.1.2. Great Clips	2
1.1.3. DMV appointment	2
1.1.4. GnomGuru	2
1.2. Comparison of Existing Applications	3
2. NARRATIVE	4
2.1. Problem Statement	4
2.2. Product Description	4
2.3. Product Scope	5
2.3.1. Scheduling.....	5
2.3.2. Delays Management.....	6
2.3.3. Priorities Management	8
2.3.4. Professor – Professor Appointments.....	10
2.3.5. Time Conflicts	11
2.3.6. Messages	11
2.3.7. Professor Leave.....	11

2.4 System Requirements.....	11
3. DEVELOPED APPLICATION DESIGN	12
3.1. Architectural Design Specifications	12
3.1.1 Firebase real-time database.....	12
3.1.2. Firebase Cloud Messaging	13
3.1.3. Android Studio.....	13
3.1.4. MultiDex	13
3.1.5. RecyclerView.....	14
3.1.6. Retrofit	14
3.2. Design Flow Diagram	15
3.3. Use Case Diagrams	16
3.4. Rescheduling Complexities	18
3.4.1. Case Study 1	19
3.4.2. Case Study 2	19
3.4.3. Case Study 3	20
3.4.4. Case Study 4	20
3.4.5. Case Study 5	21
3.4.6. Case Study 6	21
3.4.7. Case Study 7	22
3.4.8. Case Study 8	22
3.4.9. Case Study 9	23
3.4.10. Case Study 10	23
3.4.11. Case Study 11	24

3.4.12. Case Study 12	24
3.4.13. Case Study 13	25
3.4.14. Case Study 14	25
3.4.15. Case Study 15	26
3.4.16. Case Study 16	26
3.4.17. Case Study 17	27
3.4.18. Case Study 18	27
3.5. User Interface.....	28
3.5.1. Student Registration and Login	28
3.5.2. Student Homepage and Menu-options.....	29
3.5.3. Requesting Appointment	30
3.5.4. Professor Registration and Login.....	31
3.5.5. Professor's Profile.....	32
3.5.6. Professor's Appointment Requests	33
3.5.7. Appointment Confirmation.....	34
4. IMPLEMENTATION OF APPLICATION MODULES	35
4.1. Firebase Connectivity	35
4.2. User Permissions.....	35
4.3. Passwords Encryption.....	36
4.4. Configuration Key	37
4.5. Google Services	37
4.6. MultiDex	38
4.7. Notifications from Firebase	38

4.8. Check Available Time	39
4.9. Book Appointment.....	39
4.10. Notifications to Firebase	40
4.11. Single Value Event Listener	41
4.12. Cancel and Reschedule	41
4.13. Delay Modification	42
5. TESTING AND EVALUATION	44
5.1. Launching Application.....	44
5.2. Positive and Negative Test Cases	45
5.3. Requesting Appointment Testing	46
5.4. Requesting during Non-Office Hours	47
5.5. Notifications.....	48
5.6. Professor taking Leave.....	49
5.7. Student Starting Appointment.....	50
5.8. Student Starting Negative Test Case.....	51
5.9. Handling Priorities	52
5.10. Rescheduling due to Delay	53
5.11. Delay Handling	54
5.12. Other Test Cases	55
5.13. User's Evaluation.....	57
6. CONCLUSION AND FUTURE WORK	58
BIBLIOGRAPHY and REFERENCES	59
APPENDIX A.....	60

LIST OF FIGURES

Figure 2.1. Algorithm for delay management.....	7
Figure 2.2. Flow diagram for delay management	7
Figure 2.3. Flow diagram for priority management.....	9
Figure 2.4. Algorithm for priority management.....	10
Figure 3.1. Architecture for the application.....	12
Figure 3.2. Design flow diagram.....	15
Figure 3.3. Use case diagram for student.....	16
Figure 3.4. Use case diagram for professor.....	17
Figure 3.5. Delay handling case 1.....	19
Figure 3.6. Delay handling case 2.....	19
Figure 3.7. Delay handling case 3.....	20
Figure 3.8. Delay handling case 4.....	20
Figure 3.9. Delay handling case 5.....	21
Figure 3.10. Delay handling case 6.....	21
Figure 3.11. Priority handling case 1.....	22
Figure 3.12. Priority handling case 2.....	22
Figure 3.13. Priority handling case 3.....	23
Figure 3.14. Priority handling case 4.....	23
Figure 3.15. Priority handling case 5.....	24
Figure 3.16. Priority handling case 6.....	24
Figure 3.17. Priority handling case 7.....	25
Figure 3.18. Priority handling case 8.....	25

Figure 3.19. Priority handling case 9.....	26
Figure 3.20. Priority handling case 10.....	26
Figure 3.21. Priority handling case 11.....	27
Figure 3.22. Priority handling case 12.....	27
Figure 3.23. Student registration and login	28
Figure 3.24. Student's homepage and menu-options.....	29
Figure 3.25. Student requesting appointment.....	30
Figure 3.26. Professor login and menu-options.....	31
Figure 3.27. Updating professor's profile (office & free hours)	32
Figure 3.28. Professor's appointment request and message sending.....	33
Figure 3.29. Professor's appointment request and confirmation.....	34
Figure 4.1. Firebase Connectivity.....	35
Figure 4.2. User Permissions.....	36
Figure 4.3. Code for encrypting user details.....	36
Figure 4.4. Base url and server key	37
Figure 4.5. Google services.....	37
Figure 4.6. MultiDex.....	38
Figure 4.7. Notifications in firebase.....	38
Figure 4.8. Checking professor's availability.....	39
Figure 4.9. Booking appointment.....	40
Figure 4.10. Sending notifications.....	40
Figure 4.11. Saving and retrieving from firebase.....	41
Figure 4.12. Cancelling and rescheduling appointment.....	42

Figure 4.13. Schedule manipulations due to delay.....	43
Figure 5.1. Launching and login screens.....	44
Figure 5.2. Successful and not successful login screens.....	45
Figure 5.3. Requesting appointment test case.....	46
Figure 5.4. Requesting appointment during non-office hours	47
Figure 5.5. Notifications and my application status.....	48
Figure 5.6. Professor taking Leave.....	49
Figure 5.7. Student starting and ending appointment.....	50
Figure 5.8. Student attempting to start before scheduled date	51
Figure 5.9. Handling priorities	52
Figure 5.10. Rescheduling due to delay.....	53
Figure 5.11. Delay handling.....	54

LIST OF TABLES

Table 1.1. Comparison of Existing Systems.....	3
Table 5.1. Test Results	55
Table 5.2. User Ratings.....	57

1. BACKGROUND AND RATIONALE

Importance of time drives every human to perform the tasks in the most effective way. It is very essential to utilize time in proper manner to complete his tasks. Unnecessary time is wasted by common situations such as being stuck in traffic or waiting in line for appointments. In most parts of the world, appointment scheduling still relies on using time logs. There are already various applications for online appointment scheduling [2]. Still the lack of proper effective implementation for the scheduling is causing a huge delay and confusion. To make an appointment, one needs to send an email or contact professor requesting for appointment and must wait for the reply. It's a waste of time for the professor to check his whole schedule for the day and create space for a new appointment. This android application "Appointment Scheduler & Manager" would be very much helpful to save the time of an individual in the above case. Thus, there is no need for a person to be waiting for somebody in front of their door to meet them as everything is already scheduled and even the changes will be notified to the user in this application.

1.1 Existing Applications

Currently, there are various scheduling applications available online, which helps people to schedule appointments. However, most of the existing applications only bother about how to schedule appointments, and don't care about managing those appointments and handling various issues or delays.

1.1.1 Set More Appointments: In this application, users can book appointments by selecting the date and time slot available. But all the time slots are prefixed to 30

minutes. This application might be inefficient in the cases where an appointment ends less than or more than the prefixed time.

1.1.2 Great Clips: This is a well-known hair saloon that has an application for scheduling appointments. The users should register their appointments by using their phone numbers. The application books the next available slot for them and shows the remaining waiting time. It happens most frequently that the user might not readily available for his appointment as the next available slot might be less than five minutes. The users who miss the time slots are manually pushed to the end of the line and some don't even show up at all.









































1.1.3 DMV appointment: This motor licensing cooperation has a SMS based appointment scheduling process. The users should register their phone number to avail the service. The users will get message based notifications about the waiting time and can reschedule or cancel time by replying to these messages. The users can request intentional delay by requesting delay time only in minutes but the granted new waiting time always depends on the other waiting users.

1.1.4 GnomGuru: In this application, customers can schedule appointments with various companies. This application only schedules appointment but does not handle any changes in the appointment. It assumes that every person arrives exactly on time without any delays, which is not practical. It doesn't notify people about any delays or updates, which ultimately leads to waiting. It also doesn't give priorities to the appointments and all the appointments are treated equally. It would be helpful if there is a way to prioritize the appointment so that the appointments with higher priorities will be handled accordingly.

1.2. Comparison of Existing applications

Table 1.1 shows what are the existing features in various other applications like setMore, GreatClips, GnomGuru, Drivers Licence and Appointment Manager. This shows how the current application “Appointment Manager” is different from all the existing applications in terms of multiple helpful features.

Table 1.1. Comparison of Existing Systems

	SetMore	Great Clips	GnomGuru	Drivers Licence	Appointment Manager
Selecting Date/Time-slots					
Notifications					
Cancel Appointments					
Delay Handling					
Flexible time duration					
Priority Handling					
Flexible available timings					
Next day rescheduling					

2. NARRATIVE

2.1 Problem Statement

There are multiple appointment scheduling applications already available in the market. But there is a need for an application which manages the appointments as well. Few cases where one feels the need for managing appointments are as follows: A set of students might have set up an appointment with the professor. Suddenly if the professor had to take an unexpected leave for a day or two, instead of sending an email to each and every person who has taken the appointments, he can just click a button in the application which automatically cancels the appointments and notifies the user. Also, if there are any unexpected delays, just a single informs the person about it instead of the person coming and waiting to meet on time without knowing about the delay.

2.2 Product Description

The purpose of Appointment Scheduler & Manager application is to automatically schedule and manage students' appointments with an employer/professor of an organization by accessing their calendar in the user requested time and duration. This application would help both the professors and students to save their time. Managing delays and priority appointments is the primary aim of the application. This application is basically implemented in two interfaces: student interface and professor interface. Two main features of the application are scheduling and managing. Below details clearly explains of how this could be achieved.

2.3 Product Scope

This application can be accessible by any student and professor who has an Android mobile with the application installed and proper network connected. The application is designed to meets the needs of both students and professors. Application handles various features which are described as follows:

2.3.1 Scheduling

A person during registration specifies if he is a student or a professor and provides his name, user id, password. If professor, he must specify his designation, office hours and any extra free hours available for the day. Once a student logs in to the application, he can open professor's profile. In the profile, one can find the name of the professor, designation, office hours, free hours and already scheduled appointment times.

By a single click on "Request Appointment" button in professor's profile, one can request an appointment with the professor by specifying the following.

- Request data and time
- Reason for the appointment
- Expected duration (From and To times)
- Mark as normal/important

The request notification will be now sent to the professor who can view the details provided by the student and can change the duration and mark as important if needed. The professor can approve or deny a request and send a message to the student id needed.

Once the professor approves the request, application checks for the following cases:

- If the request time of the student matches with the office/free hours of the professor and if there is no other appointment already scheduled at that time, then appointment will be scheduled at the given time.
- If the request time doesn't match with the office/free hours, student will be notified about it to check and change the request time.
- If there is any other scheduled appointment during the request time, application checks if the recent appointment request is normal or marked as important. If it is normal, application sends a notification to the student who sent the recent request to change his request time. If the recent request is marked as important and already scheduled one is normal, and if the request time of already scheduled one is not within 24 hours, it will be asked to change the time to give place to the priority appointment. Otherwise, i.e., if both recent and already schedules ones are of high priority, the recent will be asked to move. Thus, the application finds a perfect slot to schedule the appointment and notifies the student about it.

2.3.2. Delays Management: The algorithm implemented for handling delays in the application is represented in figure 2.1 and the flow chart is represented in figure 2.2.

Algorithm for Scheduling appointments

```

1: Compute delay = actual_endTime - scheduled_endTime.
2: If delay ≤ 5 minutes
3:   No Rescheduling
4: else
5:   If next_startTime ≤ current_time
6:     time_diff = current_time - next_appointment_startTime.
7:     next_appointment_startTime = next_appointment_startTime + time_diff.
8:     next_appointment_endTime = next_appointment_endTime + time_diff.
9:     If next_appointment_startTime & next_appointment_endTime IN office_hours
10:      send_rescheduled_notification.
11:      reschedule_next_appointment_in_list goto Line 5.
12:   else If next_appointment_startTime & next_appointment_endTime IN free_hours.
13:     send_rescheduled_notification.
14:     reschedule_next_appointment_in_list goto Line 5.
15:   else
16:     lookup_nextDay()
17:     If lookup_nextDay = 'success'
18:       send_rescheduled_notification.
19:       reschedule_next_appointment_in_list goto Line 5.
20:   else
21:     cancel_appointment.
22:     send cancel notification.

```

Figure: 2.1 Algorithm for handling delays.

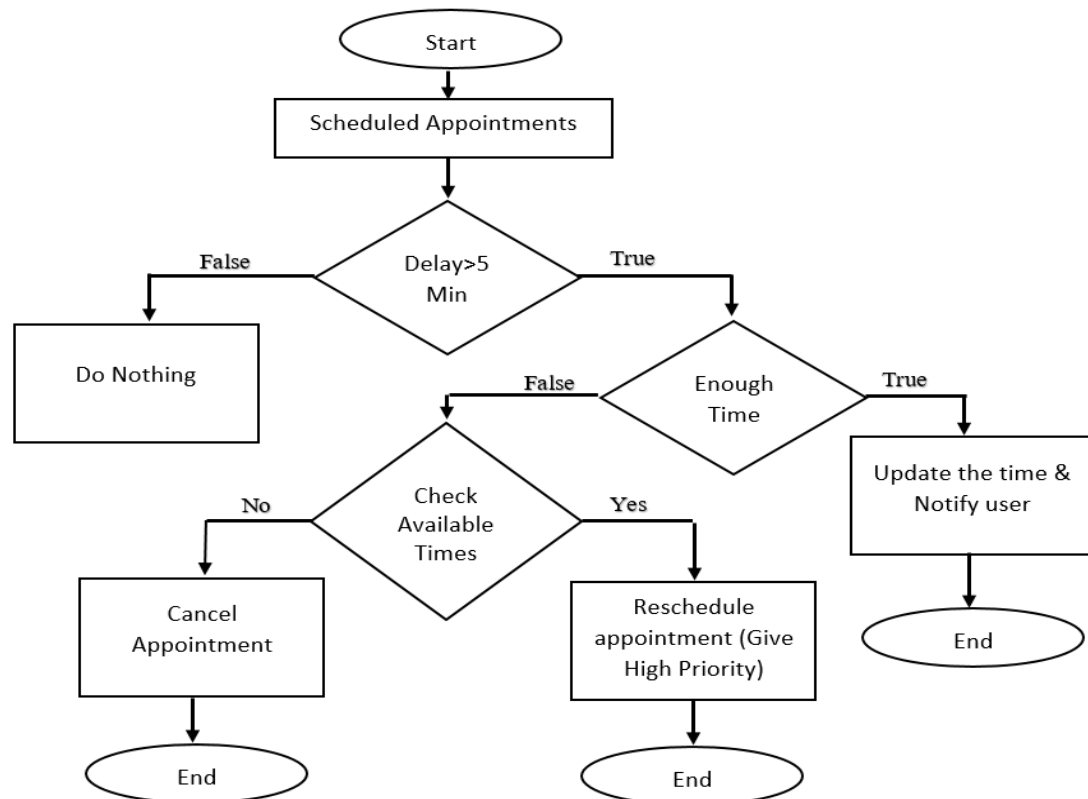


Figure 2.2. Flow Diagram for Delay Management.

Basically, every appointment can only be scheduled by giving a gap of 5 minutes before the previous appointment. As one cannot predict the exact duration for a meeting, an extra 5 minutes' time will be made available for each appointment. The student must click on the "start" button when the meeting is starting and "done" button when he is done with the meeting. Whenever a person clicks on the done button, the application checks for the scheduled end time of the appointment and compares with that of actual end time. If the difference between the actual end time and scheduled end time is greater than 5 minutes, then there is a delay. Then the application retrieves the professor's office hours, free hours and all the scheduled appointments for that day and sorts them in the order of time.

Now the application checks the difference between end time of current appointment and start time of the next appointment. If it is negative, it must be moved by adding the difference time to next appointment's start and end time. These values are now stored in a new array and same process is continued till all the appointments down the list are moved accordingly. If there is no available time for the last appointment in the list, application checks if there are available times to reschedule it. If found, it reschedules and if not, cancels the appointment.

2.3.2 Priorities Handling

Application gives priorities to the appointments in the following cases:

If the professor approves an appointment to be very important, it is given high priority. If the professor cancels an appointment after scheduling once, that appointment will be given higher priority when rescheduling for the next day. If an appointment gets cancelled for the day because of the prior appointment's delay, it will be given high priority. The idea is not to disturb the priority appointment times once they are scheduled. If the application is

changing the schedule of an appointment, its priority/weight must be increased so as not to change a single appointment multiple times. Priorities will be handled by giving weights to each of them and the higher weight one will be given higher priority compared to the lower weight ones [7].

Algorithm followed to schedule priority tasks is represented in the flow diagram as follows:

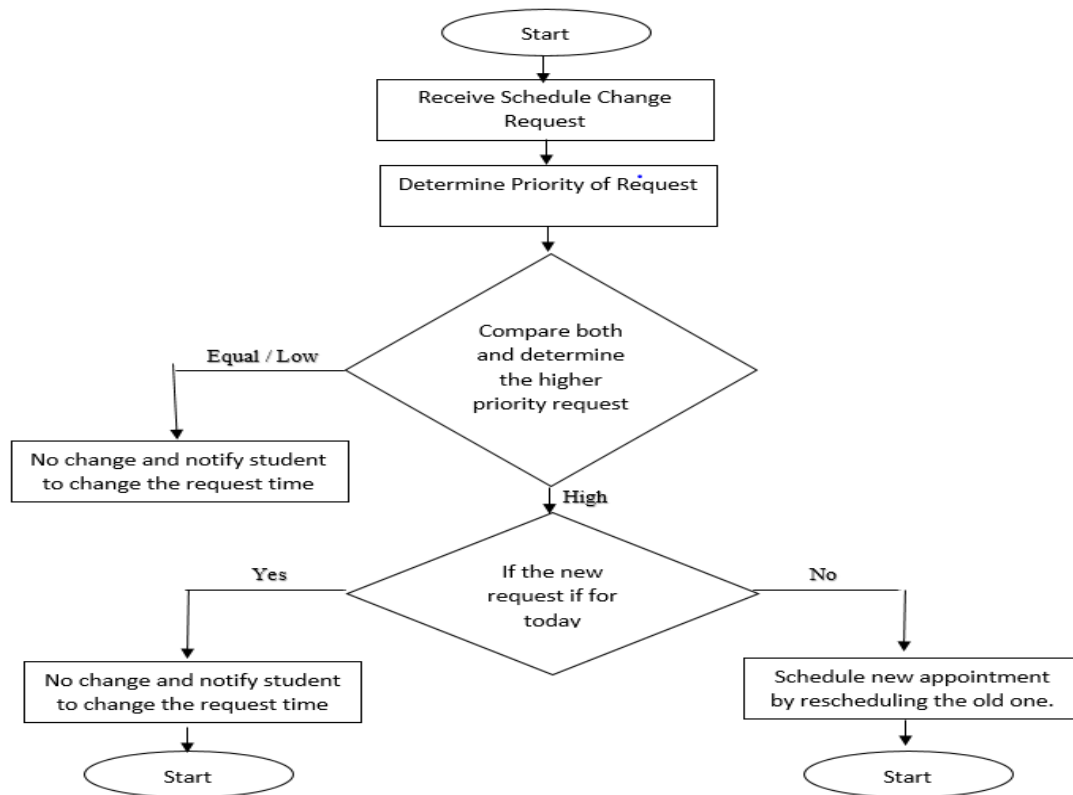


Figure 2.3. Flow Diagram for Priorities Management.

As shown in figure 2.3, if a new schedule change request arrives, the application determines the priority of the new request and then compares the priorities (weights) of both the requests [7]. If both have the same priority or the new request have the lower priority, then the schedule doesn't change and the user will be notified to schedule at some other time. If the new request has higher priority, application again checks for the

requesting start time of the new appointment. If that request for today, the schedule will not be changed and student will be notified to request an appointment again. If the new request is important and the already scheduled one is general and the appointment request is not for today, then the new appointment will be scheduled by rescheduling the old (already scheduled) appointment at some other time on the same day or next day.

Algorithm implemented to achieve priority handling is shown in figure 2.4.

Algorithm for high priority schedule request for upcoming days	
1:	while <i>scheduled_appointmentsThatDay</i> do
2:	If <i>scheduled_endTime</i> \leq <i>requestedHP_startTime</i>
3:	continue ;
4:	else if <i>scheduled_startTime</i> \geq <i>requestedHP_endTime</i>
5:	continue ;
6:	else
7:	If <i>scheduledAppointment_status</i> = 'high priority'
8:	<i>requestStatus</i> = 'rejected'
9:	<i>send cancel_notification</i>
10:	<i>discard rescheduleList[]</i>
11:	end while
12:	else
13:	<i>add to rescheduleList[]</i>
14:	continue
15:	If <i>requestStatus</i> != 'rejected'
16:	while <i>rescheduleList[]</i> do
17:	<i>lookup_nextDay()</i>
18:	If <i>lookup_nextDay</i> = 'success'
19:	<i>send_rescheduled_notification.</i>
20:	continue ;
21:	else
22:	<i>cancel_appointment.</i>
23:	<i>send cancel_notification.</i>
24:	<i>requestStatus</i> = 'accepted'
25:	<i>send rescheduled_notification</i>

Figure 2.4. Algorithm for priority handling.

2.3.3. Professor – Professor Appointments

This application also allows a professor to request an appointment with the other professor. The status of this appointment will be important by default. Though there are any existing general appointments at that time, professor-professor appointment will be

given higher priority and will not be rescheduled.

2.3.4. Time Conflicts

If a student requests a general appointment during professors working hours (non-office hours), Student will be notified about it to change the request times.

2.3.5. Messages

This application also allows the professor to send a message to the student if needed and student can also reply to it. But student cannot directly send any message to professor as it is not needed.

2.3.6. Leave

If a professor would like to take a leave for a day or two, he doesn't need to cancel all the appointments scheduled for that day individually. He can simply click the leave button which automatically cancels all the appointment for the day and notifies the users.

2.4 System Requirements

The following requirements are used to develop this application.

- Database:
 - Firebase: This is helpful to send notifications form the server and is cost effective.
- Integrated Develop Environment:
 - Android Studio: It has improved interface design compared to others and has modules which helps running, testing and debugging independently.
- Software:
 - Android Software Development Kit
 - JAVA SE 8
- Hardware:
 - A Personal Computer
 - Android Mobile Device

3. DEVELOPED APPLICATION DESIGN

Below is the architecture of the application with the details of requests and responses between the database server and users as shown in figure 3.1.

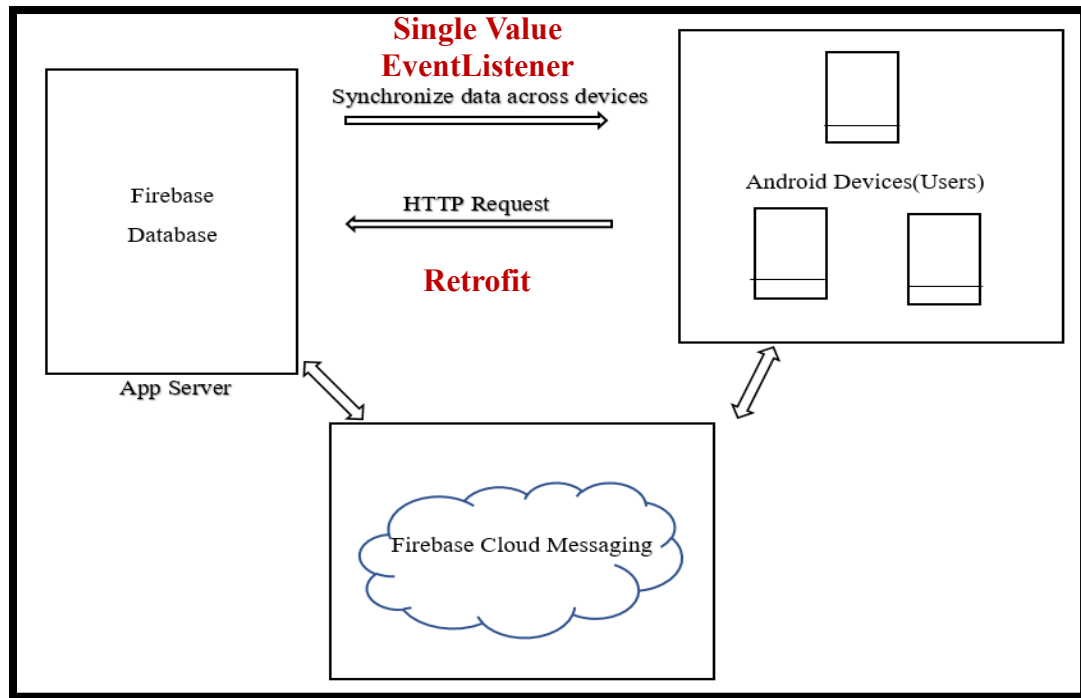


Figure 3.1. Architecture of the Application.

3.1. Architectural Design Specifications

3.1.1 Firebase Real-time Database

Firebase is a NoSQL cloud database used to store data. Data is synced across all clients in real-time, and remains available when the app goes offline. Firebase is a cloud-hosted database which processes http requests. Data is stored as JSON and synchronized in real-time to every connected client. When cross-platform apps are built with Android all the clients share one Real-time Database instance and automatically receive updates with the newest data. This application uses the feature of Firebase Cloud Messaging.

3.1.2 Firebase Cloud Messaging

Firebase Cloud Messaging (FCM) is a cross-platform messaging solution that lets the user reliably deliver messages at no cost. Using FCM, you can notify a client app that new email or other data is available to sync [5]. A person can send notification messages to drive user reengagement and retention. For use-cases such as instant messaging, a message can transfer a payload of up to 4KB to a client app.

Key Capabilities

- Send notification messages or data messages.
- Versatile message targeting.
- Send messages from client apps.

3.1.3. Android Studio

Android studio is the official integrated development platform for Android platform. It was designed based on IntelliJ IDEA [1]. The first stable build of android studio was released in December 2014 and from then it replaced Eclipse Android Development Tools (ADT) as Google's primary IDE for native android application development. We can also test the applications developed on a virtual device provided by Android Studio. Present version of Android studio has the features like Gradle-based build support, lint tools, ProGuard integration and rich layout editor which allows drag and drop of UI components etc.

3.1.4. MultiDex

MultiDex patches the application context class loader to load classes from more than one dex file. The primary classes.dex must contain the classes necessary for calling

this class methods. Secondary dex files named classes2.dex, classes3.dex... found in the application apk will be added to the class loader after first call to install.

3.1.5. RecyclerView

The RecyclerView is a new ViewGroup that is prepared to render any adapter-based view in a similar way. It is supposed to be the successor of ListView and GridView, and it can be found in the latest support-v7 version. RecyclerView widget is used when we need data collections whose elements change at runtime based on user action or network events.

3.1.6. Retrofit

It is a type-safe HTTP client for Java and Android. Exchanging data between a mobile app and a backend server is an essential need for many development projects. Retrofit aims to make the exchanging task even simpler.

3.2 Design Flow

A flowchart is a visual representation of the sequence of steps and decisions needed to perform a process. The design flow diagram of the application is shown in the figure 3.2.

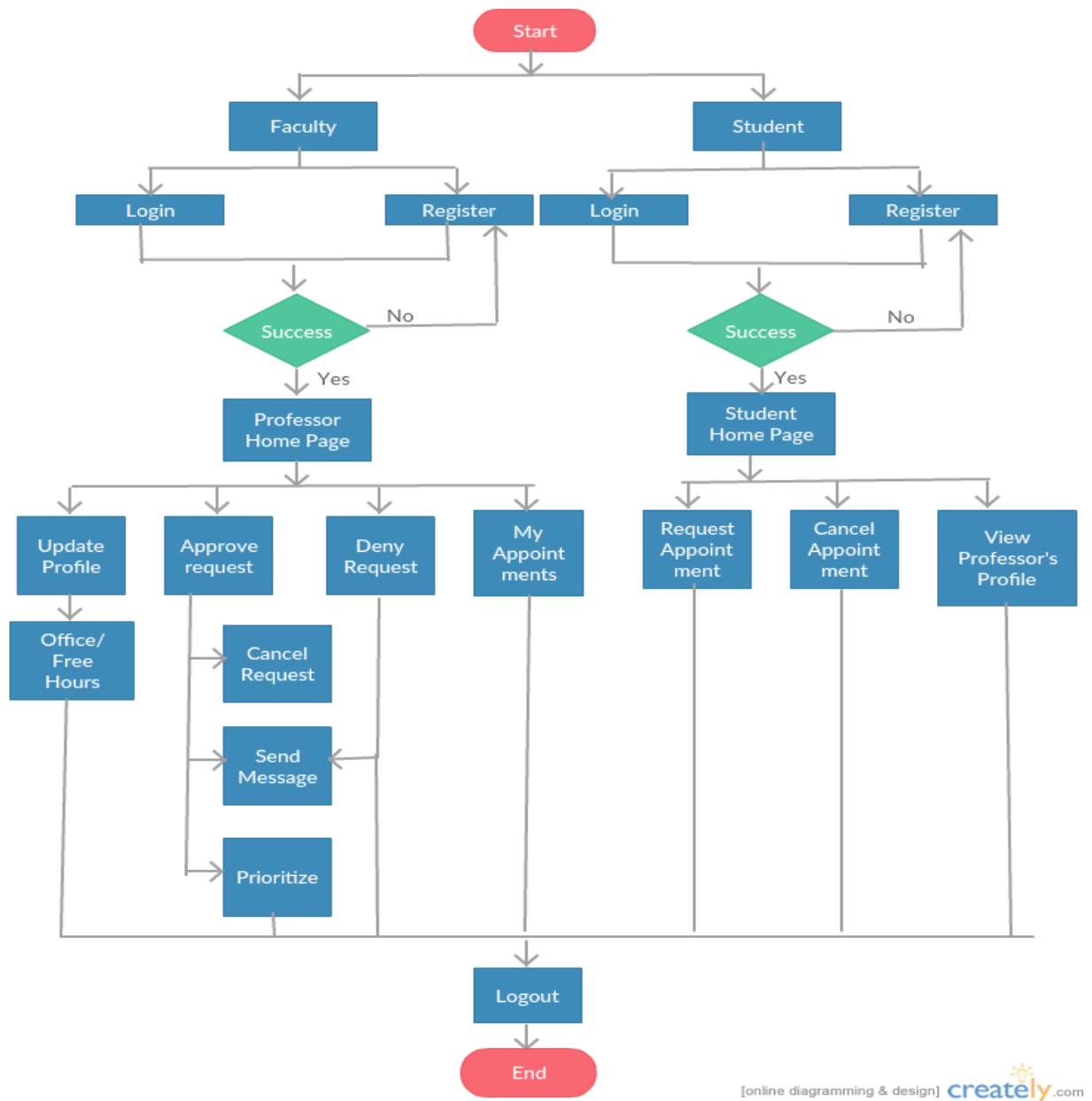


Figure 3.2. Design Flow Diagram.

Student or the faculty initially need to register and log in to the application. Once a student is successfully logged in, he will be able to check the professor's profile for office and free hours and request a new appointment or cancel the existing appointment. A professor will be able to update his office hours, request an appointment with another professor, cancel any existing appointments or prioritize the requests.

3.5. Use Case Diagrams

After registration and log in, student will be able to request both one-time appointments or weekly appointments. Once he schedules a weekly appointment, it will be scheduled throughout the semester once every week. Student can also mark an appointment to be regular or important. Student could see the office hours, extra free hours and already booked times of the professor so that he could choose the best time out of all available times. The Use case diagram for a student is shown in figure 3.3.

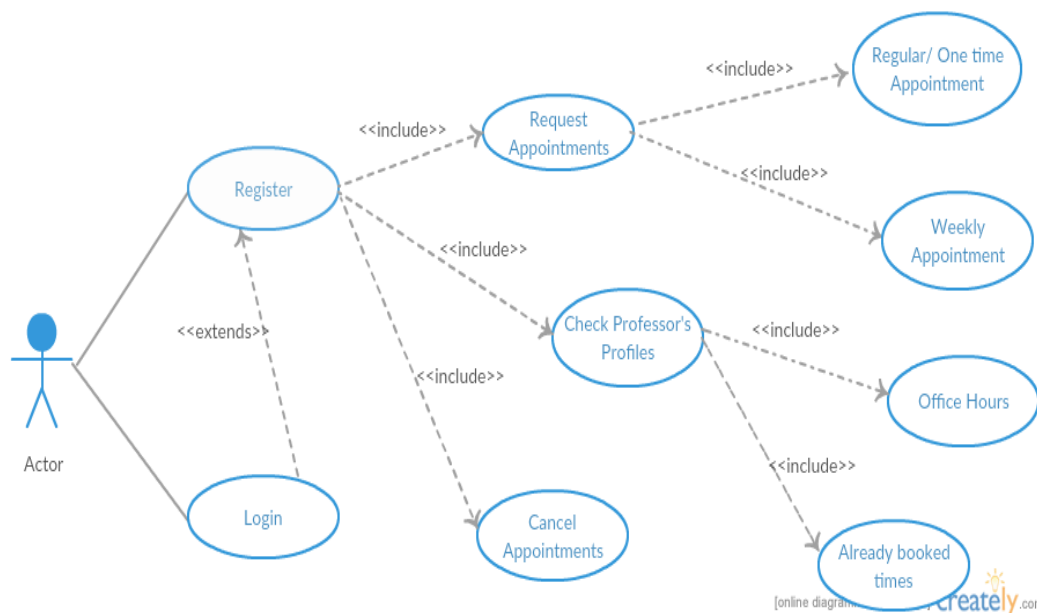


Figure 3.3. Use Case Diagram for Student.

After registration and log in, a professor can update his office hours, request a new appointment with another professor or cancel an existing appointment. Cancelling of an appointment may be due to any of the reasons: appointment might not be needed anymore or had to do some other work at that time. A professor can even request a weekly or one-time appointment with another professor like students. The use case diagram for a professor in the application is shown as in figure 3.4.

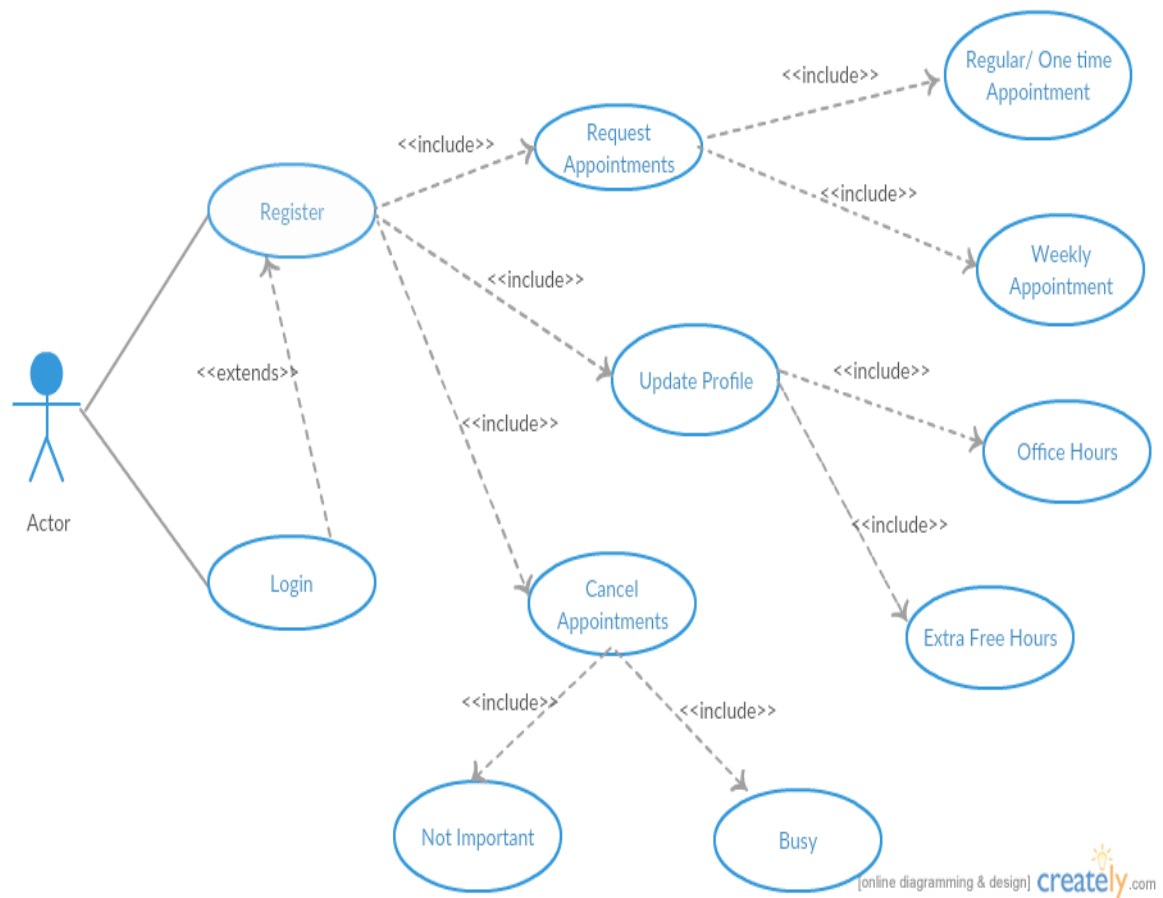


Figure 3.4. Use Case Diagram for Professor.

3.6. Rescheduling Complexities

The current strategies for the rescheduling appointments solely lies on just pushing all the appointments till the end of the day. They still fail to acknowledge the various complexities involved in the process of rescheduling. The various types of the complexities that are involved and their handling is represented in the following cases studies:

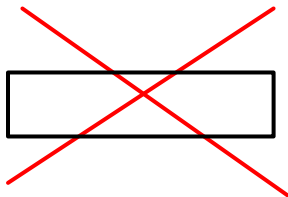
Legend:



➔ Allotted Slot (GA – General Appointment, HP – High Priority appointment)



➔ To be Rescheduled for the next day



➔ Appointment request Cancelled



➔ Available Hours



➔ High Priority Request

High Priority



➔ Delayed Appointment

delay

3.6.1. Case Study 1: If there is a delay in first appointment as shown in figure 3.5, rest of the appointments will be pushed accordingly as there is more available time.

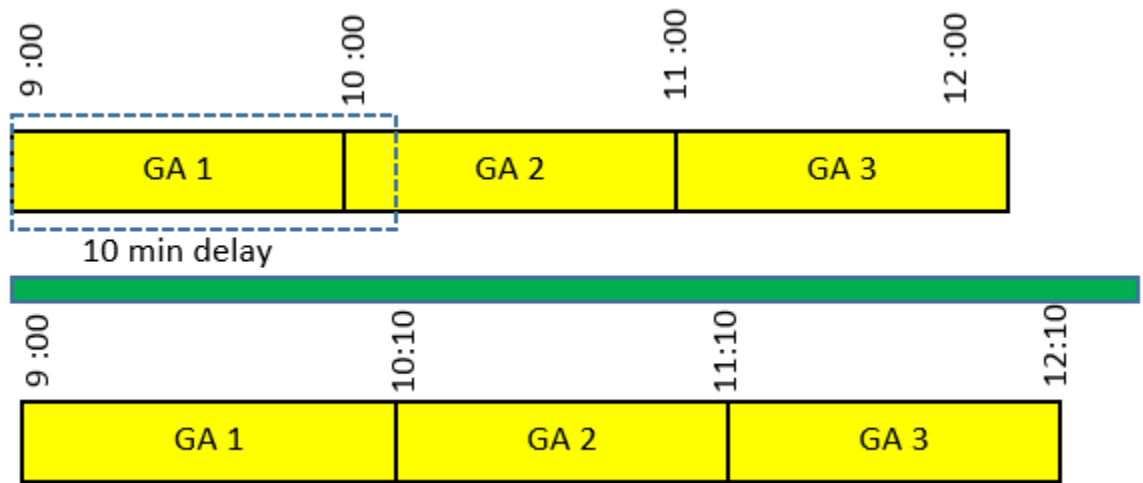


Figure 3.5. Delay handling case 1.

3.6.2. Case Study 2: In this case, if there is a delay in first appointment, only the second appointment will be pushed accordingly as it is facing the impact of delay while third appointment remains the same as shown in figure 3.6.

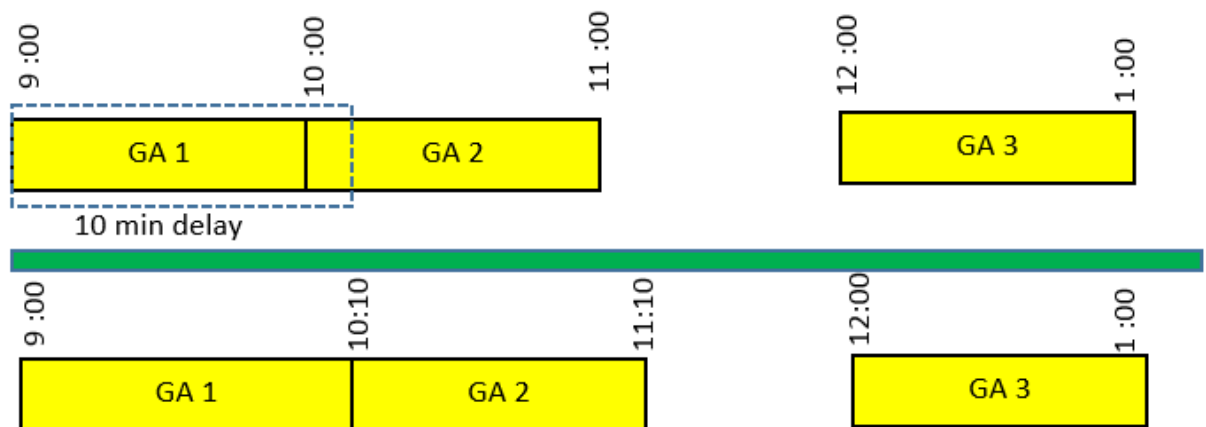


Figure 3.6. Delay handling case 2.

3.6.3. Case Study 3: In this case as shown in figure 3.7 , if there is a delay in first appointment, second one can be pushed accordingly. But the third appointment cannot be pushed as it exceeds professor's available time. So, the application tries to reschedule it to the next day by finding appropriate slot. If there is no available slot, it will be cancelled and student will be notified.

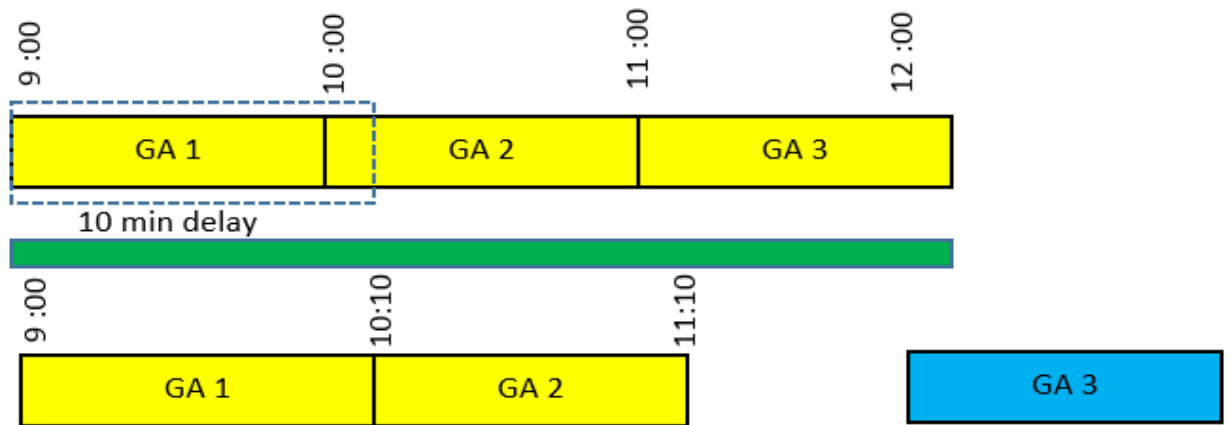


Figure 3.7. Delay handling case 3.

3.6.4. Case Study 4: Due to the 10-minute delay in the first appointment, second appointment must be pushed by 10 minutes. But as there is no available time, application reschedules it to next day as there is no available slots today as shown in figure 3.8.

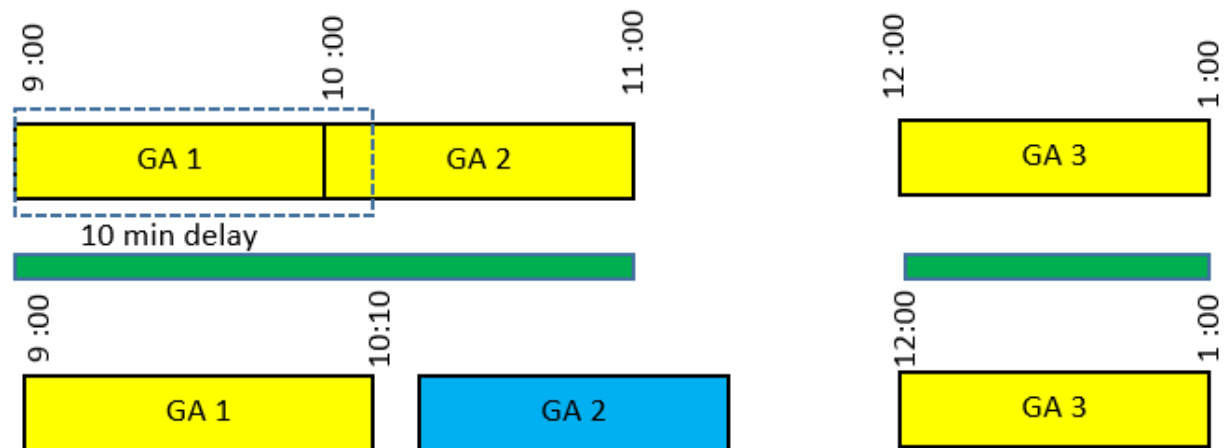


Figure 3.8. Delay handling case 4.

3.6.5. Case Study 5: Due to the 10-minute delay in the first appointment, second appointment must be pushed by 10 minutes. But as there is no available time, application checks if there are any available slots today and reschedules accordingly as in figure 3.9.

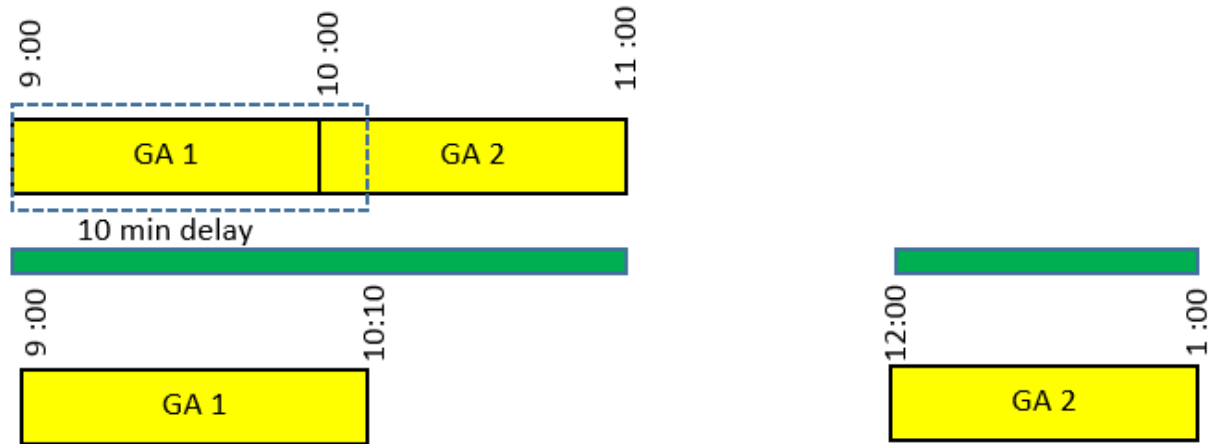


Figure 3.9. Delay handling case 5.

3.6.6. Case Study 6: In this case as shown in figure 3.10, the delay of the first appointment exceeds the duration of second appointment. There is no possible room to push second and third appointments, so they will be rescheduled for the next day if there is available time.

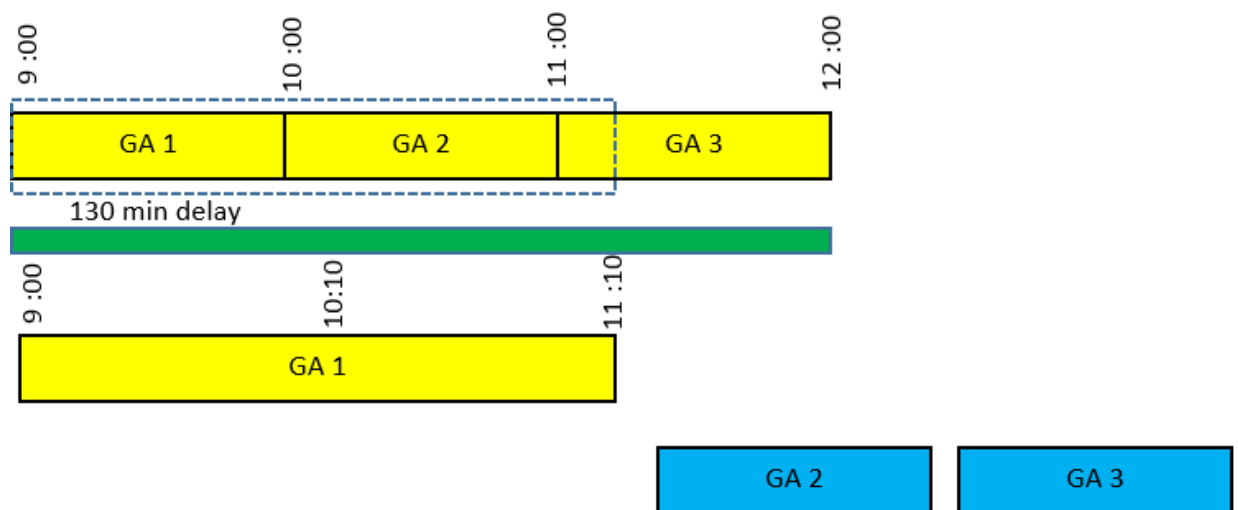


Figure 3.10. Delay handling case 6.

3.6.7. Case Study 7: In this case, the first general appointment is rescheduled as the same slot is requested by a high priority appointment as shown in figure 3.11.

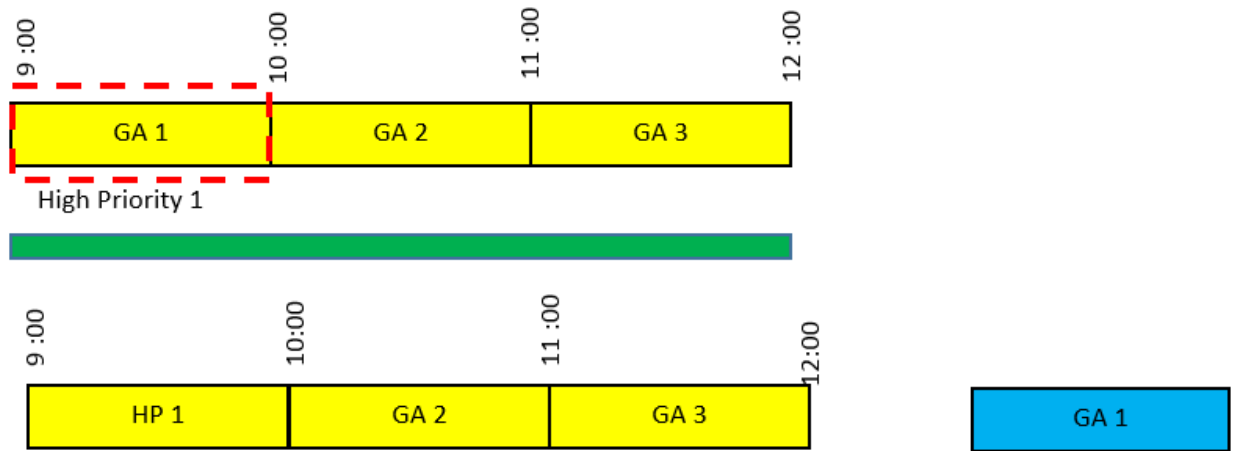


Figure 3.11. Priority handling case 1.

3.6.8. Case Study 8: General appointment 1 will be rescheduled to the next day as it is facing the impact of newly arrived high priority appointment request as shown in figure 3.12.

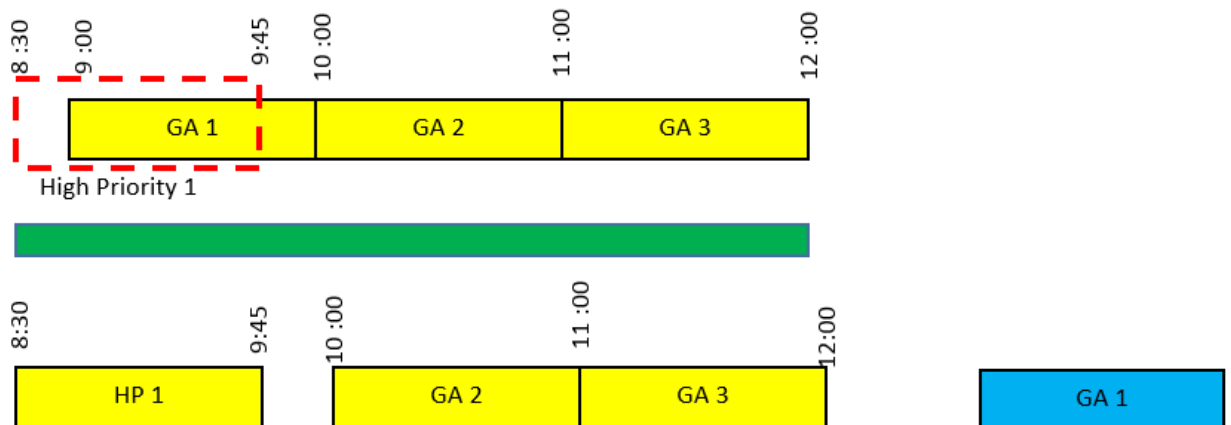


Figure 3.12. Priority handling case 2.

3.6.9. Case Study 9: Both the general appointments 1 and 2 will be rescheduled to the next day as they are facing the impact of newly arrived high priority appointment request as shown in figure 3.13.

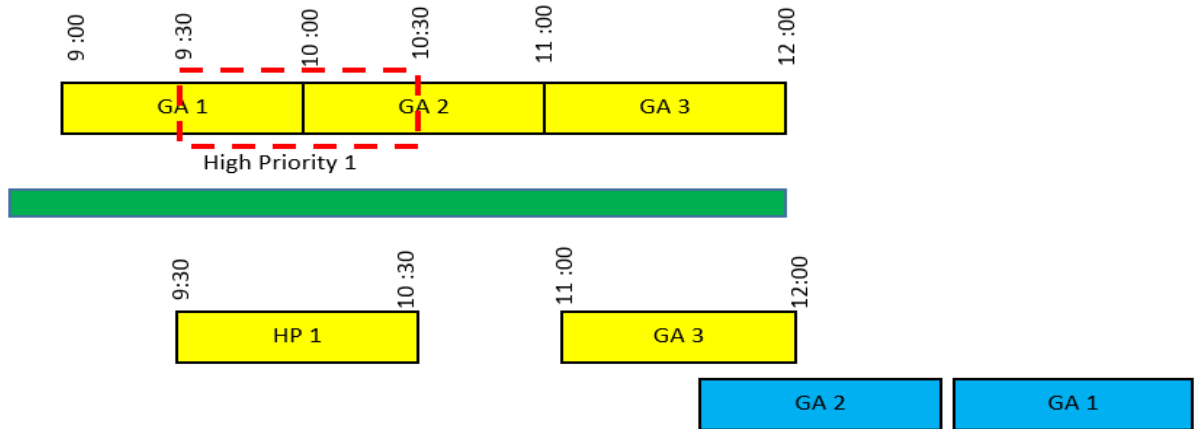


Figure 3.13. Priority handling case 3.

3.6.10. Case Study 10: General appointment 2 will be rescheduled to the next day as it is facing the impact of newly arrived high priority appointment request as shown in figure 3.14.

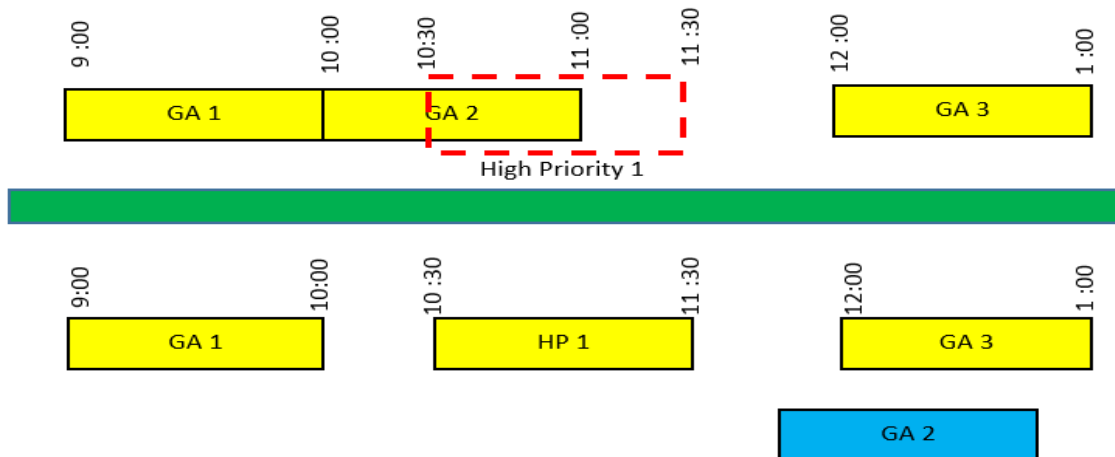


Figure 3.14. Priority handling case 4.

3.6.11. Case Study 11: General appointment 2 will be rescheduled to the next day as it is facing the impact of newly arrived high priority appointment request as shown in figure 3.15.

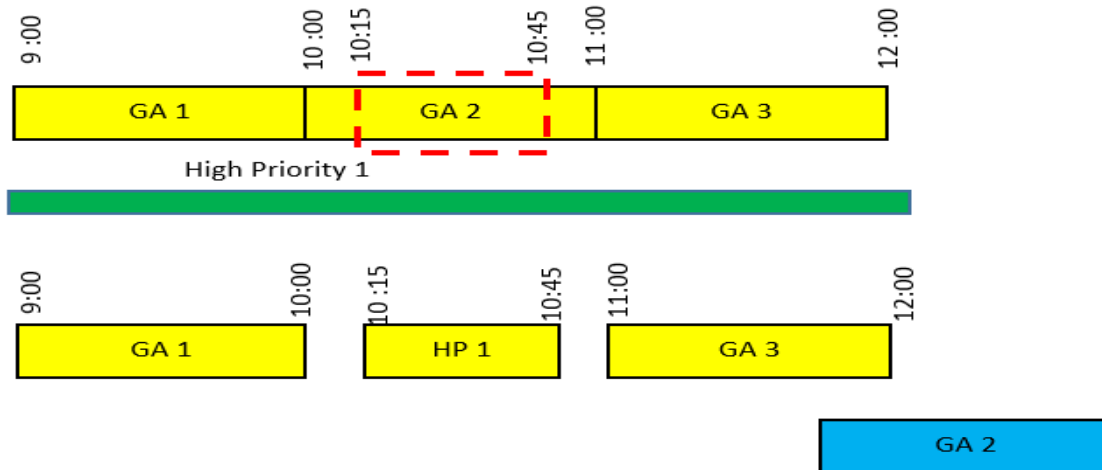


Figure 3.15. Priority handling case 5.

3.6.12. Case Study 12: General appointment 2 will be rescheduled to the next day as it is facing the impact of newly arrived high priority appointment request as shown in figure 3.16.

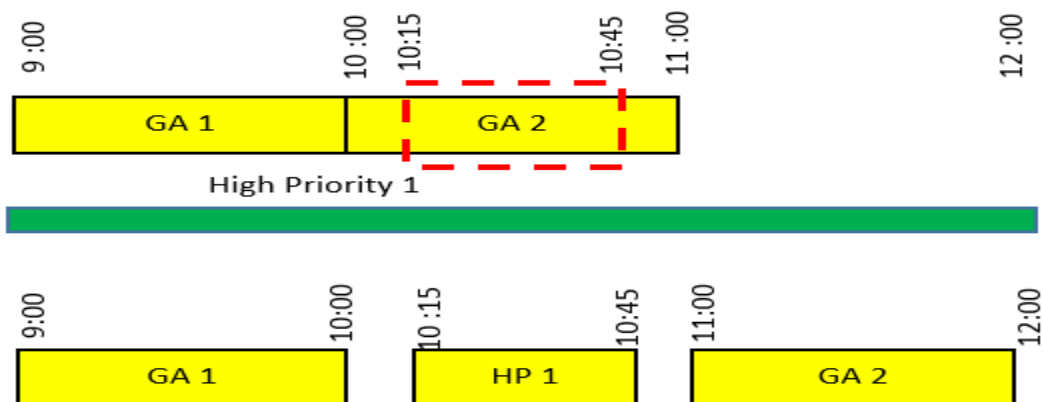


Figure 3.16. Priority handling case 6.

3.6.13. Case Study 13: As shown in figure 3.17, the problem is when the high priority appointment is scheduled as the last appointment of the day. In case of any delay in the previous appointments, this high priority appointment will be rescheduled to the next day. The application cannot handle this case as it is using array lists and handling all appointments one by one.

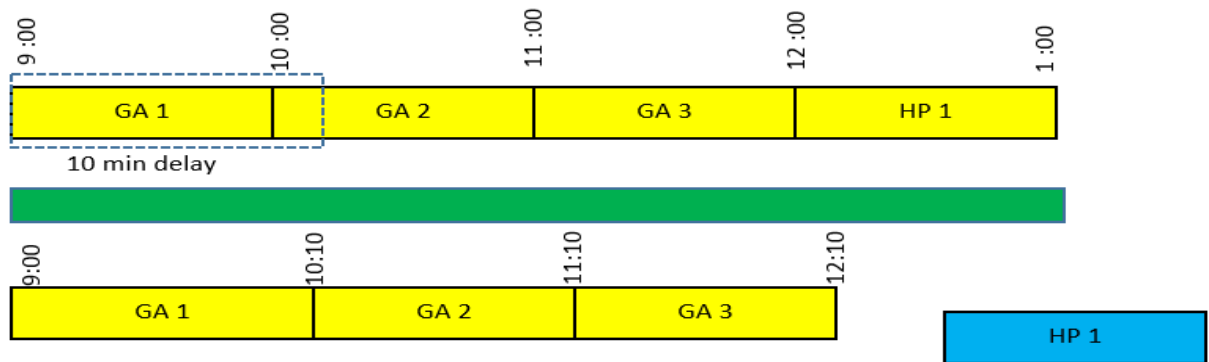


Figure 3.17. Priority handling case 7.

3.6.14. Case Study 14: As shown in figure 3.18, if a new high priority appointment (HP 2) is requested for the same slot of already scheduled high priority appointment (HP 1). The new request (HP 2) will be cancelled and the application notifies the user to request for some other available slot.

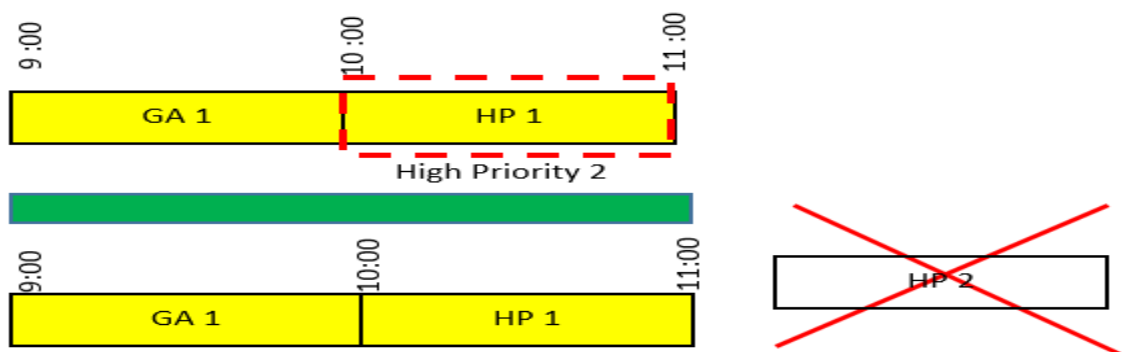


Figure 3.18. Priority handling case 8.

3.6.15. Case Study 15: If a new high priority request impacts both general and other high priority appointments as shown in figure, the new request will be cancelled and the user will be notified as shown in figure 3.19 .

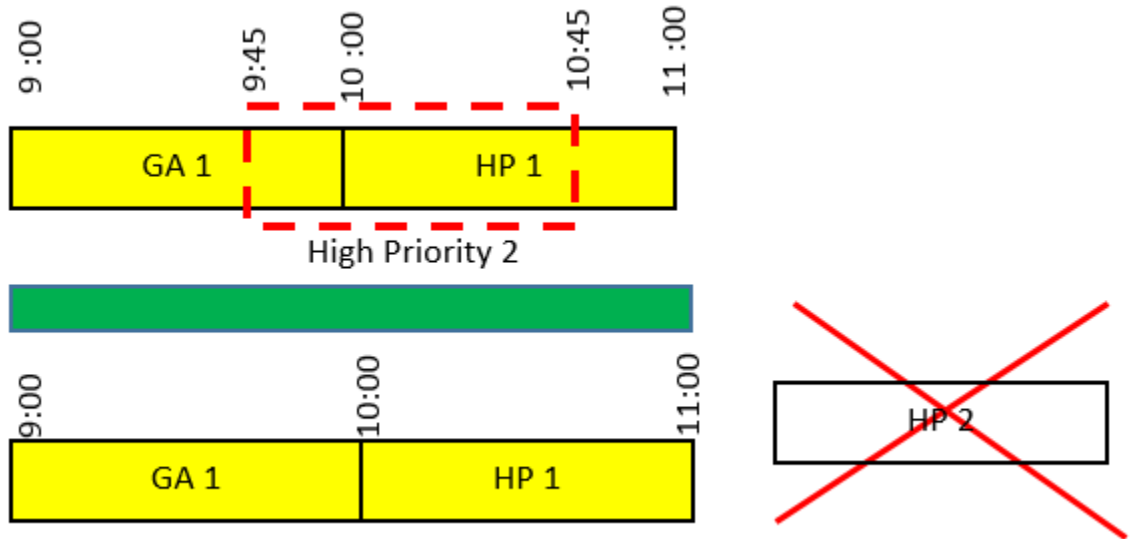


Figure 3.19. Priority handling case 9.

3.6.16. Case Study 16: If a new high priority request impacts both general and other high priority appointments as shown in figure, the new request will be cancelled and the user will be notified as shown in figure 2.30.

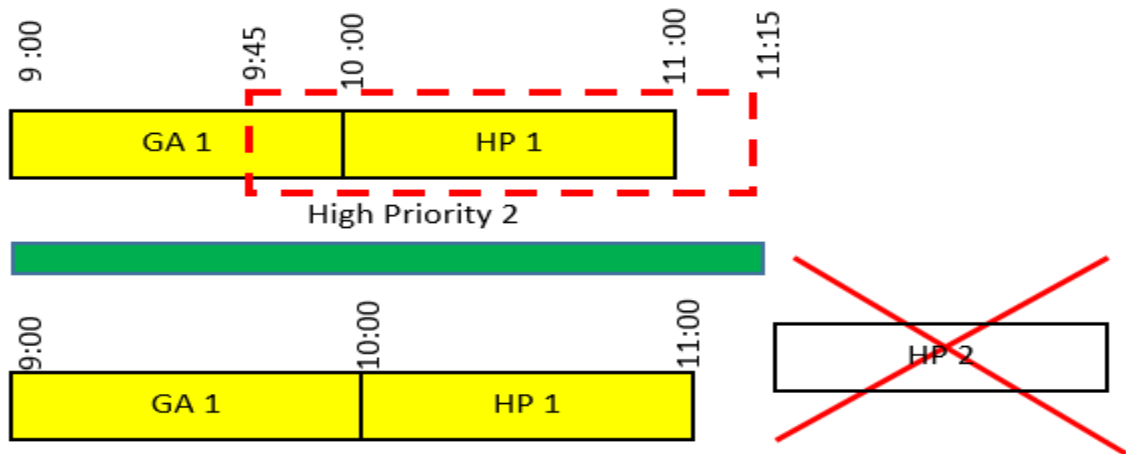


Figure 3.20. Priority handling case 10.

3.6.17. Case Study 17: As shown in figure 3.21, if a new high priority request impacts the other high priority appointment as shown in figure, the new request will be cancelled and the user will be notified.

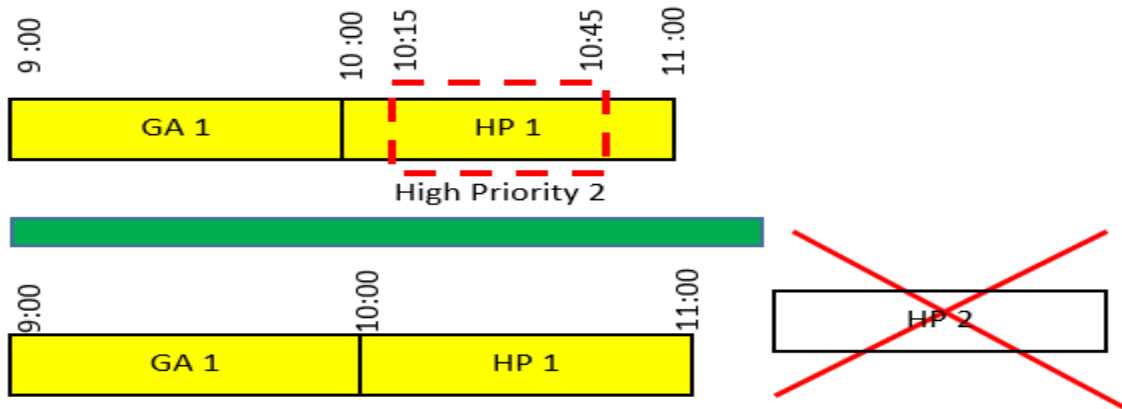


Figure 3.21. Priority handling case 11.

3.6.18. Case Study 18: If a new high priority request impacts two general appointments as shown in figure, the new request will be approved by rescheduling two general appointments for the next day as shown in figure 3.22.

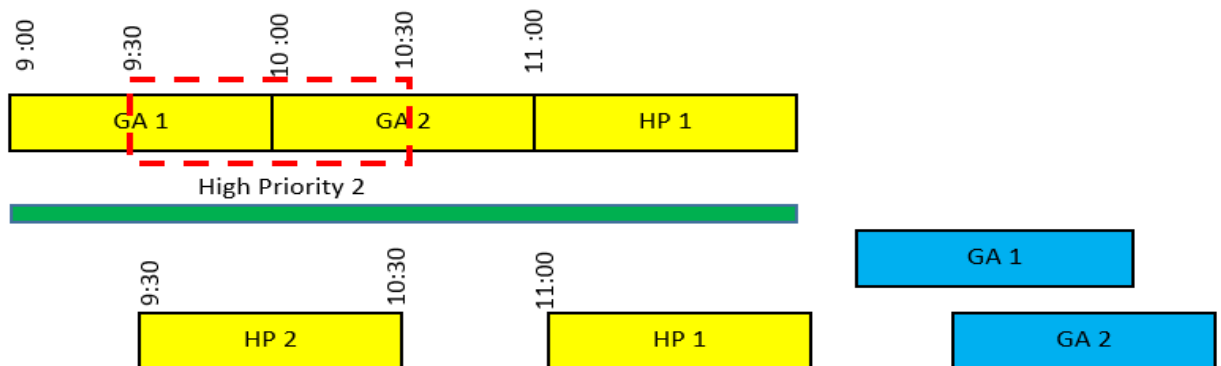


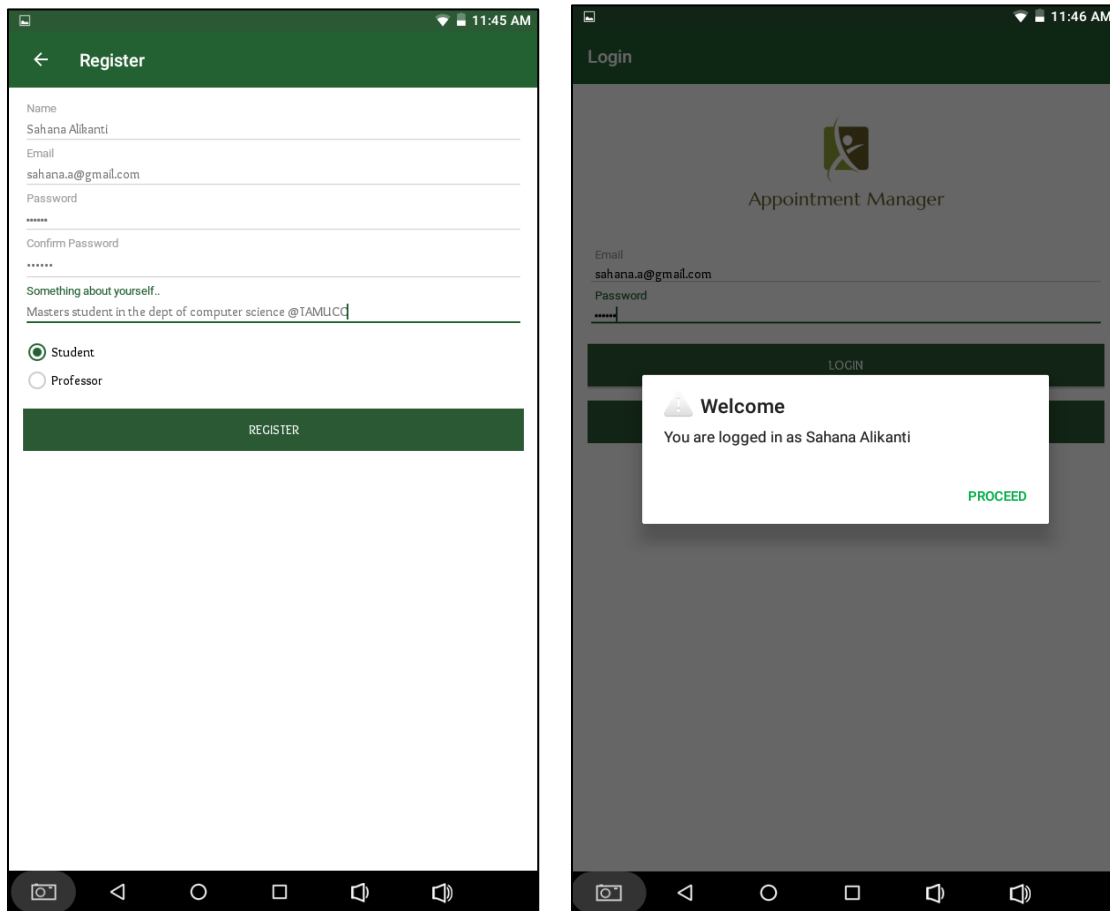
Figure 3.22. Priority handling case 12.

3.7. User Interface

This section gives the overview of user interface of the application. User interface is where user interacts with the application.

3.7.1. Student Registration and Login Screens

As a student, one must specify name, email, password and something about the self for registration. For login, a student can just type in the email id specified and matching password as shown in figure 3.23.



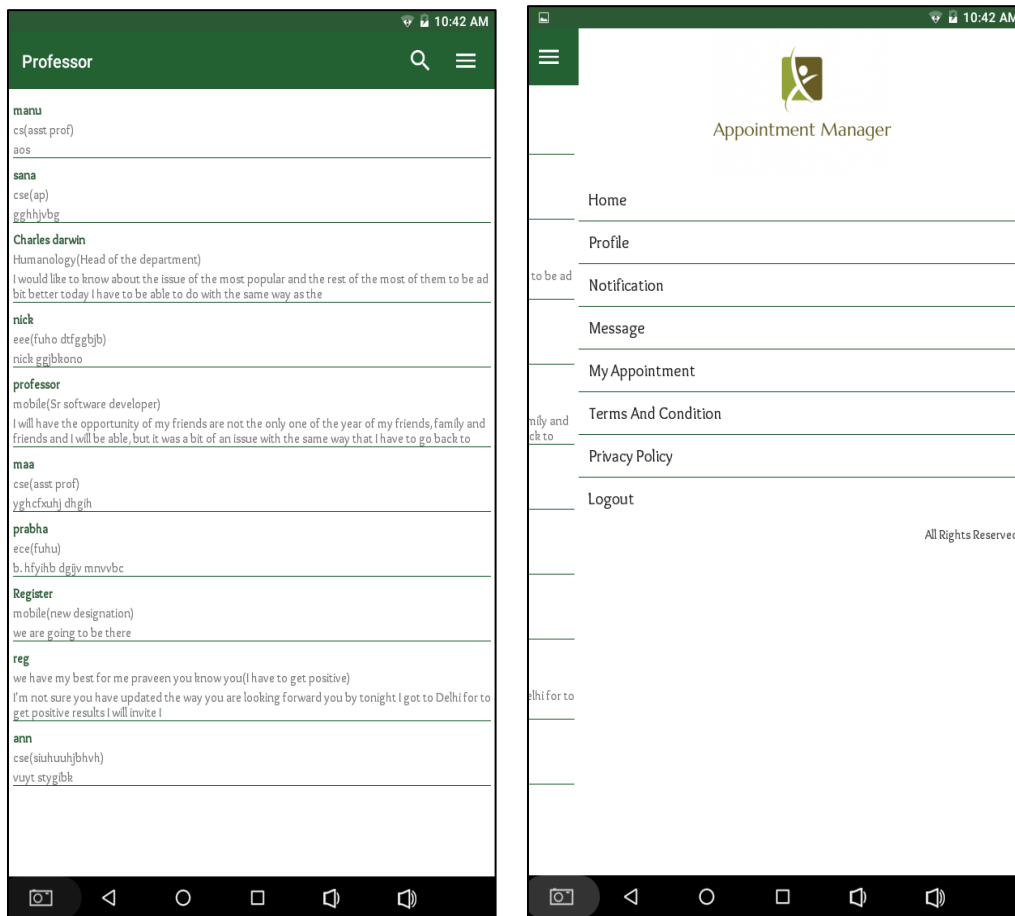
(a)

(b)

Figure 3.23. Student Registration and Login.

3.7.2. Student Homepage and Menu-options

Figure 3.24 shows the home page of a student displaying all available professor's names and a student can search for a professor in the search bar on the top of the home page. A student has menu with multiple options like Home: To display the home page, Profile: To open his/her profile, Notifications: To see all the notifications, Message: To view his/her messages, My Appointment: To see his/her appointments and status, Logout: To logout from anywhere in the application.



(a)

(b)

Figure 3.24. Student home page and Menu-options.

3.7.3. Requesting Appointment

Student initially opens a professor's profile page and clicks of 'request appointment' to request a new appointment. As in figure 3.25(c), student can request an appointment by specifying the following details. As in figures 3.25(b) student can also check for the professor's office, free hours and already booked hours.

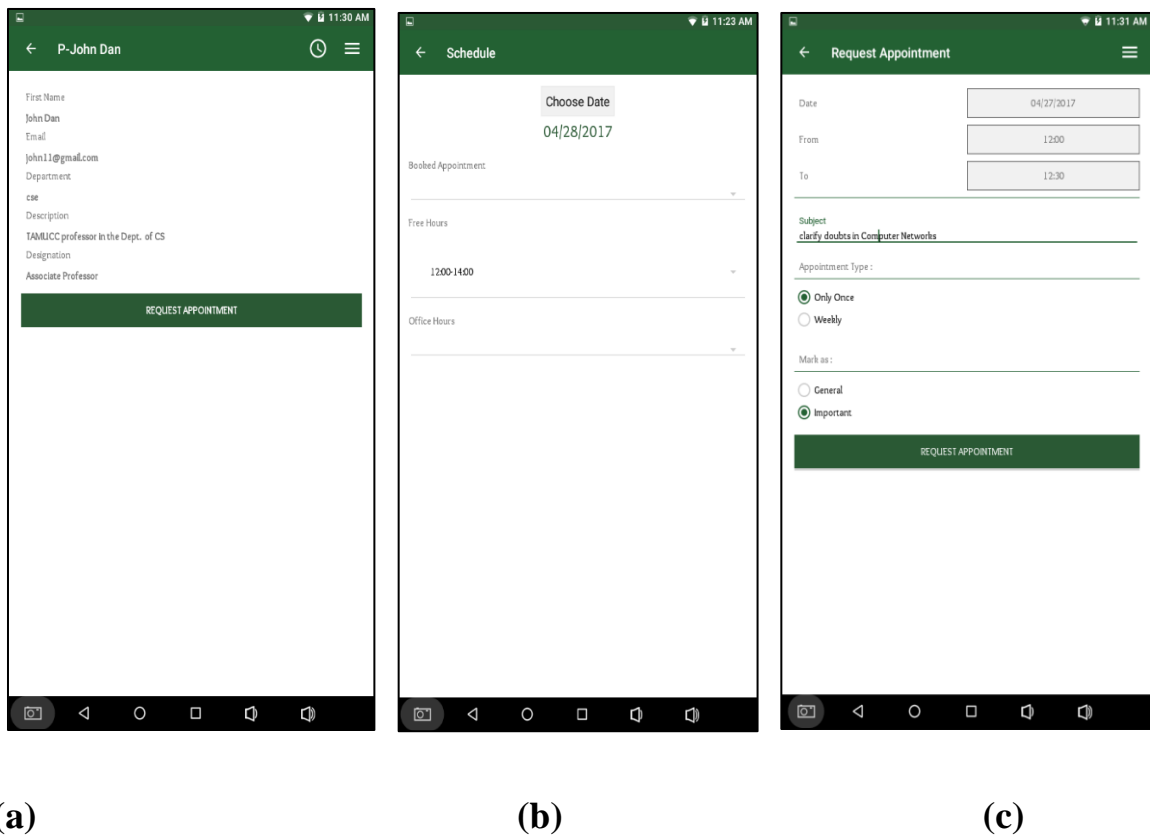


Figure 3.25. Student Requesting Appointment.

3.7.4. Professor Registration and Login

Figure 3.26 is the login screen for the professor. Once logged in, a dialogue box pops up to remind the user to update his office or extra free hours in his profile if any. A professor also has a menu with options like home, profile, notifications, messages, my appointments

and logout like that of student. But professor has an extra request appointment option to request an appointment with another professor.

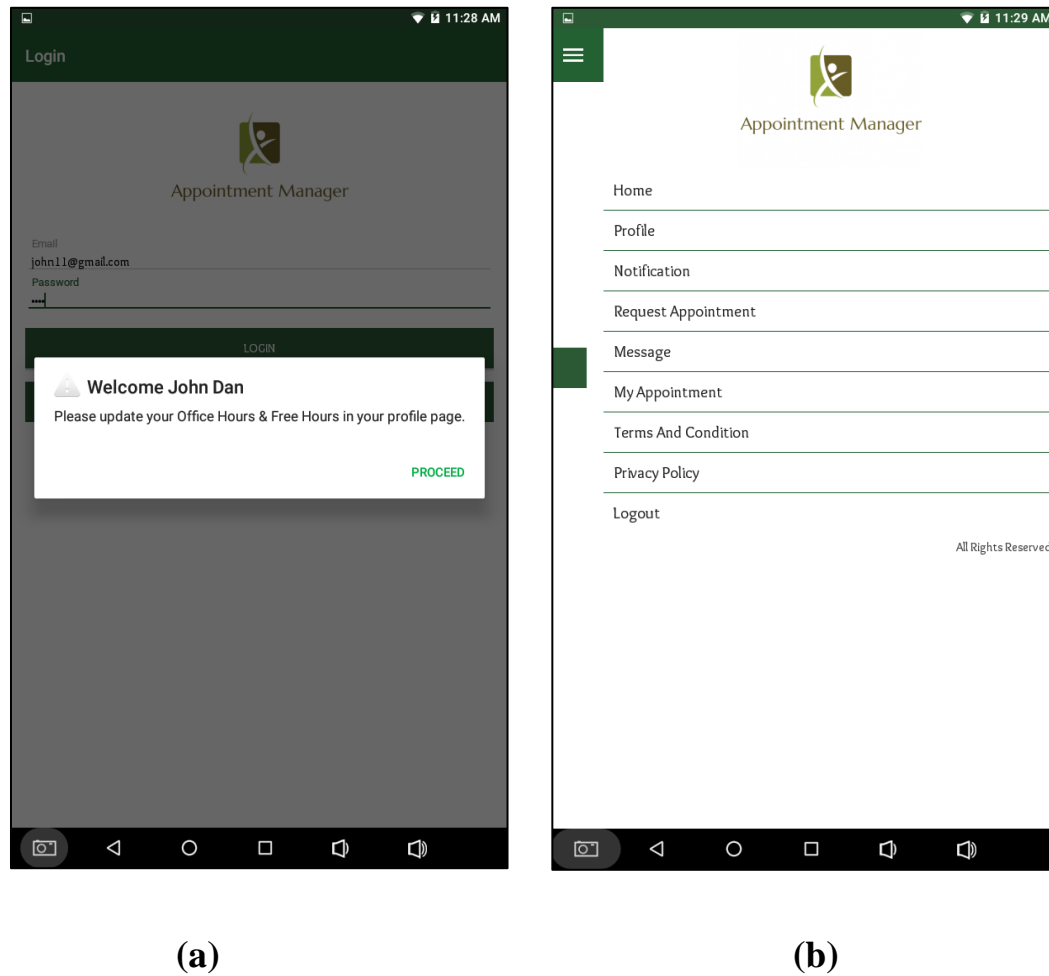


Figure 3.26. Professor Login and Menu options.

3.7.5. Professor's profile

A professor can update the office/free hours in his profile page by clicking on the 'My Schedule' button. As in figure 3.27, professor can just select a day and specify the hours for that day.

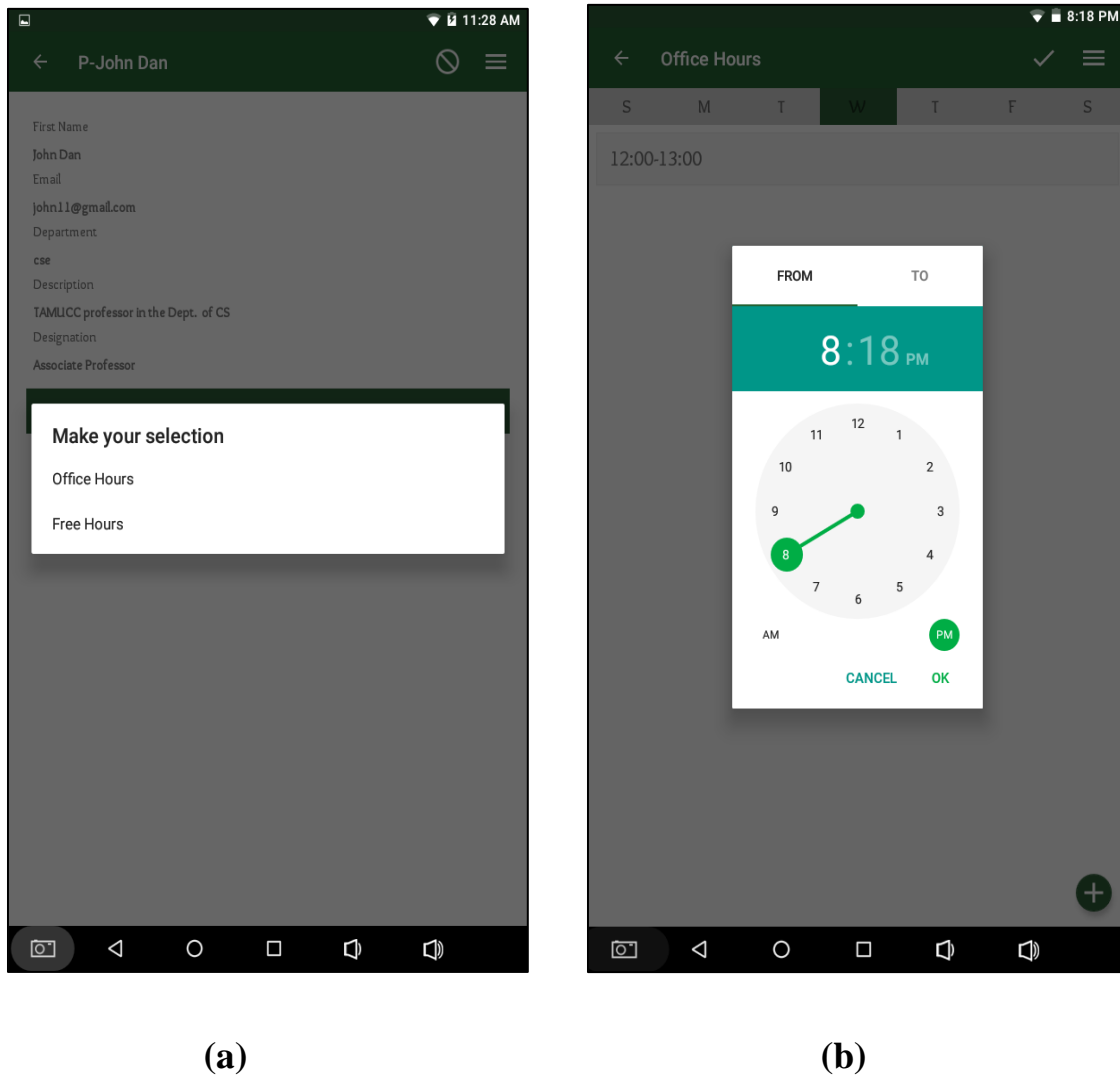


Figure 3.27. Updating professor's profile (office and free hours) using Time Picker.

3.7.6. Professor's Appointment Requests

As in figure 3.28, a professor gets the appointment requests in his home page. If he clicks on that request, he will be able to see the students details along with the duration and purpose of the appointment requested. A professor can send a message to any of the student by clicking the message icon on the top right side.

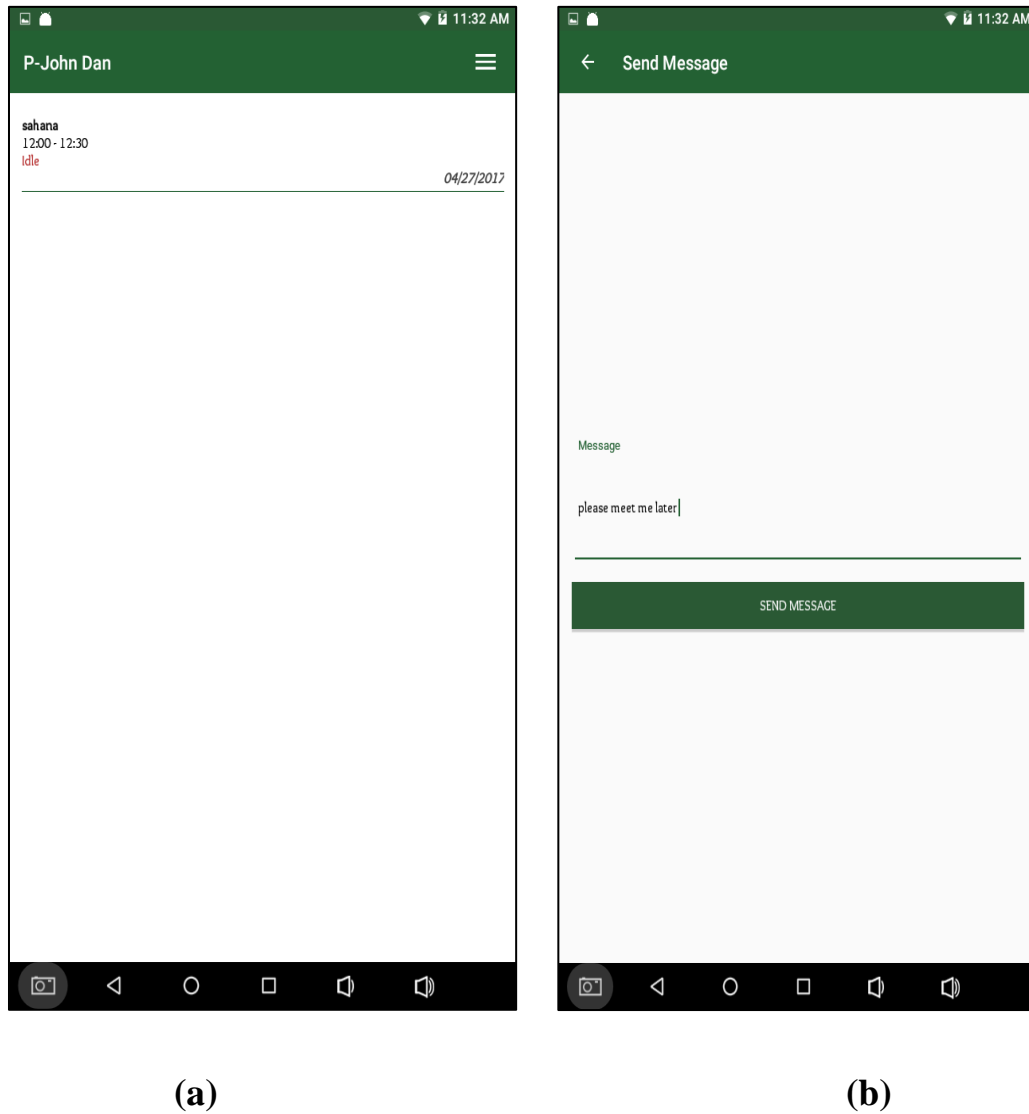
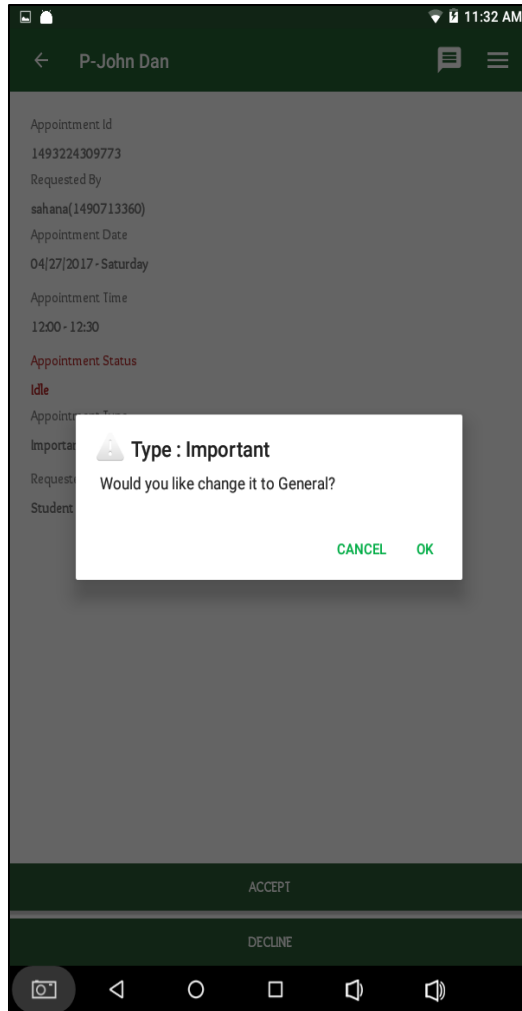


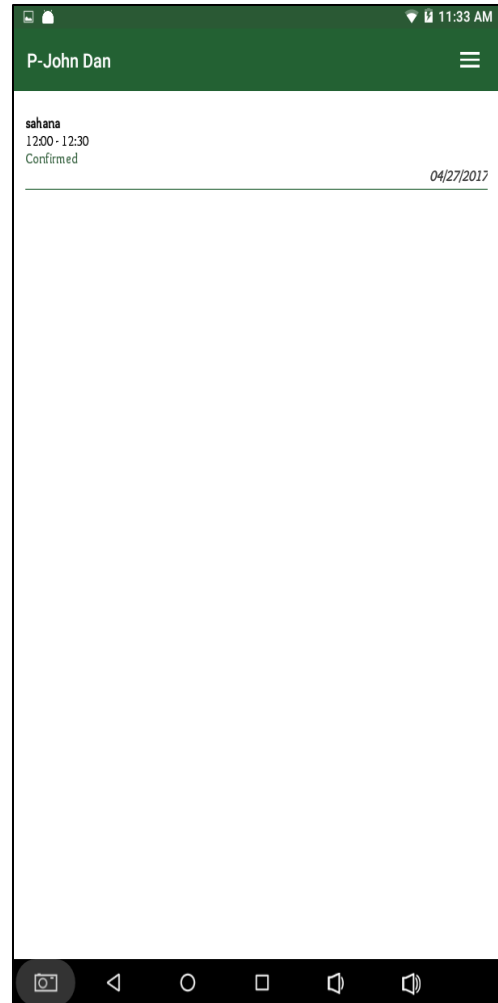
Figure 3.28. Professor's Appointment Requests and Message Sending.

3.7.7. Appointment Confirmation

Professor can change the type specifies by the student (from 'important' to 'general' or vice-versa) and either approve or decline his request. Once the request is confirmed, professor can see the status as confirmed in his home page.



(a)



(b)

Figure 3.29. Professor's Appointment Request Confirmation.

4. IMPLEMENTATION OF APPLICATION MODULES:

4.1. Firebase Connectivity

Initially when a student or professor registers their account in the application, their details must be stored in the database. To save all the details of each user, a unique id is generated for each person and details are saved accordingly. Figure 4.1 shows how these details are stored. A new database reference is also created to make it easy to store/retrieve the details from the database.

```
private void saveNewUser() {
    try {
        showProgress();
        final String email = edtEmail.getText().toString();
        final String id = String.valueOf(System.currentTimeMillis() / 1000);
        final DataUser dataUser = new DataUser();

        dataUser.setUserEmail(email);
        dataUser.setUsername(edtName.getText().toString());
        dataUser.setUserPassword(edtPassword.getText().toString());
        dataUser.setUserDescription(edtDescription.getText().toString());
        dataUser.setUserId(id);
        dataUser.setUserToken(Constant.FIREBASE_TOKEN);
        dataUser.setUserType(radProfessor.isChecked() ? Constant.PROFESSOR : Constant.STUDENT);

        final FirebaseDatabase database = FirebaseDatabase.getInstance();
        final DatabaseReference userRef = database.getReference(Constant.TABLE_USER);
```

Figure 4.1. Firebase connectivity.

4.2. User Permissions

AndroidManifest.xml is a must file for every android application. All the permissions using in the application should mention in this file. AndroidManifest.xml contains all the essential information about application. Figure 4.2 displays the permissions of Appointment Manager. Both location and internet permissions are required for this application.

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.sah.appt">

    <uses-permission android:name="android.permission.ACCESS_COARSE_LOCATION" />
    <uses-permission android:name="android.permission.ACCESS_FINE_LOCATION" />
    <uses-permission android:name="android.permission.INTERNET" />
    <uses-permission android:name="android.permission.RECEIVE_BOOT_COMPLETED" />
```

Figure 4.2. Permissions in AndroidManifest.xml.

4.3. Passwords Encryption

During registration, a person will definitely give various personal details. To protect the privacy of an individual, the sensitive data of an individual like passwords are stored into the database in encrypted format by using obscured shared preferences which uses DES algorithm. This protects the data even from the admin of the application. Figure 4.3 shows how encryption and decryption of data is done.

```
protected String encrypt(String value) {
    try {
        final byte[] bytes = value != null ? value.getBytes(UTF8) : new byte[0];
        SecretKeyFactory keyFactory = SecretKeyFactory.getInstance("PBEWithMD5AndDES");
        SecretKey key = keyFactory.generateSecret(new PBEKeySpec(SEKRIT));
        Cipher pbeCipher = Cipher.getInstance("PBEWithMD5AndDES");
        pbeCipher.init(Cipher.ENCRYPT_MODE, key, new PBEParameterSpec(Settings.Secure.getString(context, "PBEParams")));
        return new String(Base64.encode(pbeCipher.doFinal(bytes), Base64.NO_WRAP), UTF8);
    } catch (Exception e) {
        throw new RuntimeException(e);
    }
}

protected String decrypt(String value) {
    try {
        final byte[] bytes = value != null ? Base64.decode(value, Base64.DEFAULT) : new byte[0];
        SecretKeyFactory keyFactory = SecretKeyFactory.getInstance("PBEWithMD5AndDES");
        SecretKey key = keyFactory.generateSecret(new PBEKeySpec(SEKRIT));
        Cipher pbeCipher = Cipher.getInstance("PBEWithMD5AndDES");
        pbeCipher.init(Cipher.DECRYPT_MODE, key, new PBEParameterSpec(Settings.Secure.getString(context, "PBEParams")));
        return new String(pbeCipher.doFinal(bytes), UTF8);
    } catch (Exception e) {
        throw new RuntimeException(e);
    }
}
```

Figure 4.3. Code for encrypting user details

4.4. Configuration Key

As shown in figure 4.4, the base-url and key would be used for sending notifications. The key will be obtained from the firebase console.

```
public class Config {

    public static final String BASE_URL = "https://fcm.googleapis.com/";
    public static final String FCM_SERVER_KEY = "key=AAAAAGF1z4k:APA91bE5c3fJMuzx-miuHhLXzp4JLCrmkG0G49acMUmqa97Bttr";
    public static final boolean IS_LOCATION_APP = false;

}
```

Figure 4.4. Base url and server key.

4.5. Google Services

When an application is created in the google console, it generates a file as shown in figure with all the information about the project like project number, id and key values to connect database and the server. This file must be downloaded and added to the app for proper communications with the database as shown in figure 4.5.

```
{
  "project_info": {
    "project_number": "190612623241",
    "firebase_url": "https://appointment-244be.firebaseio.com",
    "project_id": "appointment-244be",
    "storage_bucket": "appointment-244be.appspot.com"
  },
  "client": [
    {
      "client_info": {
        "mobilesdk_app_id": "1:190612623241:android:f51637210580f354",
        "android_client_info": {
          "package_name": "com.sah.appt"
        }
      },
      "oauth_client": [
        {
          "client_id": "190612623241-8uiubn2fadimhk2tq0hji0qvc68ioltt.apps.googleusercontent.com",
          "client_type": 1,
          "android_info": {
            "package_name": "com.sah.appt",
            "certificate_hash": "c43aee363e0717cd4ff7ae25c5c2f82941818d72"
          }
        },
        {
          "client_id": "190612623241-921ehsk0eb17ebm2b2ikbm3r9n9pkk7t.apps.googleusercontent.com",
          "client_type": 3
        }
      ]
    }
  ]
}
```

Figure 4.5. Google services.

4.6. MultiDex

MultiDex is installed to allow the project to work in various android versions even below 5.0. This splits up the apk so as to work in different versions as in figure 4.6.

```
public class BookAppointment extends MultiDexApplication {

    @Override
    public void onCreate() {
        super.onCreate();
        MultiDex.install(this);
    }
}
```

Figure 4.6. MultiDex.

4.7. Notifications from Firebase

Notifications play a major role in Appointment Manager application. Every single change in the application creates a notification to the user, however these are helpful. Figure 4.7 shows how a notification is created and stored to the firebase so that it could be called from the server whenever required. Sending a notification to a person requires his identity (name, id) along with the notification message and the identity of the person from which it is generating.

```
@Override
public void onMessageReceived(RemoteMessage remoteMessage) {
    // TODO(developer): Handle FCM messages here.
    // Not getting messages here? See why this may be: https://goo.gl/39bRNJ

    Log.d(TAG, "remoteMessage.toString() : " + remoteMessage.toString());

    if (remoteMessage.getData().size() > 0) {
        Log.d(TAG, "Message data payload: " + remoteMessage.getData().get("message"));
        Log.d(TAG, "Message data payload: " + remoteMessage.getData().get("type"));
        mCustom = remoteMessage.getData().get("message");
        mType = remoteMessage.getData().get("type");
        sendNotification();
    }

    if (remoteMessage.getNotification() != null) {
        Log.d(TAG, "Message Notification Body: " + remoteMessage.getNotification().getBody());
    }
}
```

Figure 4.7. Notifications in Firebase.

4.8. Check Available Time

Whenever a student requests an appointment with the professor, application checks for the professor's available timings by checking his office hours, free hours and already booked appointments. If the requested time of the student is within the office/free hours of the professor and if no other person has already booked an appointment at that time, then a new appointment request will be sent to the professor.

Figure 4.8 shows how this works.

```
private void checkAvailability() {

    float requestedStartTime = Float.parseFloat(txtFromTime.getText().toString().replace(":", "."));
    float requestedEndTime = Float.parseFloat(txtToTime.getText().toString().replace(":", "."));

    int size = appointmentList.size();
    boolean isAppointment = false;

    if (size == 0)
        checkFreeTime();
    else {
        for (int index = 0; index < size; index++) {

            float existingStartTime = Float.parseFloat(appointmentList.get(index).getStartTime().replace(":", "."));
            float existingEndTime = Float.parseFloat(appointmentList.get(index).getEndTime().replace(":", "."));

            if ((requestedStartTime <= existingStartTime && requestedEndTime >= existingStartTime) ||
                (requestedStartTime <= existingEndTime && requestedEndTime >= existingEndTime)) {
                isAppointment = false;
                Utils.displayToast("This time slot is already scheduled, Please choose another time.", BookAppointment.this);
                return;
            } else
                isAppointment = true;
        }
    }
    if (isAppointment)
        checkFreeTime();
}
```

Figure 4.8. Checking Professor's Availability.

4.9. Book Appointment

To book a new appointment, the application checks the timestamp requested calls both the methods `bookAppointment()` and `recursiveBooking()` as shown in the figure 4.9.

```

private void bookAppointment() {
    if (radOnlyOnce.isChecked()) {
        DateFormat dateFormat = new SimpleDateFormat("MM/dd/yyyy HH:mm:ss", Locale.US);
        Calendar c = Calendar.getInstance();
        String timestamp = dateFormat.format(c.getTime());
        bookAppointment(txtDate.getText().toString(), timestamp);
    } else
        recursiveBooking();
}

```

Figure 4.9. Booking Appointment.

4.10. Notification to Firebase

Retrofit is a type-safe HTTP client for android and java. It integrates the data pushes all at a time from the application to the server. It is used to push all the details like student id, name, notification message and type etc. to the server at once.

```

public SendNotification(String receiverType, String requesterToken, String notificationText) {

    try {
        Retrofit retrofit = null;
        OkHttpClient.Builder builder = new OkHttpClient().newBuilder();
        builder.readTimeout(60, TimeUnit.SECONDS);
        builder.connectTimeout(60, TimeUnit.SECONDS);

        HttpLoggingInterceptor interceptor = new HttpLoggingInterceptor();
        interceptor.setLevel(HttpLoggingInterceptor.Level.BODY);
        builder.addInterceptor(interceptor);

        // builder.addInterceptor(new UnauthorisedInterceptor(context));
        OkHttpClient client = builder.build();
        retrofit = new Retrofit.Builder().baseUrl(Config.BASE_URL)
            .client(client)
            .addConverterFactory(GsonConverterFactory.create())
            .build();

        HttpRequest httpRequest = retrofit.create(HttpRequest.class);
        Call<DataNotificationResult> call = httpRequest
            .sendNotification(new DataNotify(requesterToken, new DataNotifyMsg(notificationText, receiverType)
            call.enqueue(new Callback<DataNotificationResult>() {
                @Override
                public void onResponse(Response<DataNotificationResult> response) {
                    // Utils.debug("Response " + response.body().toString());
                }
            })
    }
}

```

Figure 4.10. Sending Notifications.

4.11. Single Value Event Listener

Whenever there is a change in professor's schedule, the value event calls the single value event listener which updates the changed value to the database. This data can be retrieved from the database using dataSnapshot object as in figure 4.11.

```
private void getNotificationList() {

    try {
        showProgress();
        FirebaseDatabase database = FirebaseDatabase.getInstance();
        final DatabaseReference profRef = database.getReference(Constant.TABLE_NOTIFICATIONS);

        profRef.addListenerForSingleValueEvent(new ValueEventListener() {
            @Override
            public void onDataChange(DataSnapshot dataSnapshot) {
                stopProgress();
                if (dataSnapshot.getChildrenCount() == 0) {
                    showSnackBar(parentPanel, getString(R.string.no_data_found));
                } else {
                    for (DataSnapshot snapshot : dataSnapshot.getChildren()) {
                        DataNotification obj = snapshot.getValue(DataNotification.class);
                        if (userId.equals(obj.getReceiverId()))
                            notificationList.add(obj);
                    }
                    Utils.debug("Arraylist size : : " + notificationList.size());
                }
                adapter = new NotificationAdapter(notificationList, MyNotification.this);
                recyclerProfessorList.setAdapter(adapter);
            }
        });
    }
}
```

Figure 4.11. Saving and retrieving from firebase.

4.12. Cancel and Reschedule

If a general appointment is already scheduled and a new important request has arrived for the same time, the important appointment will be given higher priority and the general appointment will be rescheduled by increasing its priority so that it will not be rescheduled again as shown in figure 4.12.

```

private boolean cancelAndReschedule(int cancelIndex) {

    Utils.debug("Cancel and Reschedule..");

    cancelExistingAppointment(appointmentList.get(cancelIndex));

    boolean canBookAppointment = false;
    float requestedStartTime = Float.parseFloat(fromTime);
    float requestedEndTime = Float.parseFloat(toTime);

    int size = appointmentList.size();

    for (int index = 0; index < size; index++) {

        float existingStartTime = Float.parseFloat(appointmentList.get(index).getStartTime().replace(":", " "));
        float existingEndTime = Float.parseFloat(appointmentList.get(index).getEndTime().replace(":", " "));

        if ((requestedStartTime <= existingStartTime && requestedEndTime >= existingStartTime) ||
            (requestedStartTime <= existingEndTime && requestedEndTime >= existingEndTime)) {
            if (appointmentList.get(index).getAppointmentType().equals(APPT_TYPE_IMPORTANT)) {
            }
            else
                canBookAppointment = true;
        }
    }
    return canBookAppointment;
}

```

Figure 4.12. Cancelling and Rescheduling Appointment.

4.13. Delays Manipulation

If there is a delay in one appointment, the next appointments would be rescheduled based on the delay time of the prior appointment. If there is no sufficient time for the last appointment, it will be canceled for the day and rescheduled to some other day. Figure 4.13 displays how an appointment is cancelled and rescheduled to the other day.

```

private void manipulateTodayAppointment() {
    Utils.debug("manipulateTodayAppointments : ");
    Collections.sort(appointmentList, new Comparator<DataAppointment>() {
        @Override
        public int compare(DataAppointment dataAppointment, DataAppointment dataAppointment2) {
            Utils.debug(dataAppointment.getStartTime());
            return dataAppointment.getStartTime().compareTo(dataAppointment2.getStartTime());
        }
    });

    try {
        SimpleDateFormat simpleDateFormat = new SimpleDateFormat("HH:mm", Locale.US);
        String todayDate = simpleDateFormat.format(Calendar.getInstance().getTime());
        Date startTime = simpleDateFormat.parse(todayDate);
        int size = appointmentList.size();
        Utils.debug("size : " + size);
        for (int index = 0; index < size; index++) {
            if (appointmentList.get(index).getAppointmentStatus().equals(APPOINTMENT_STARTED)) {
                Date endTime = simpleDateFormat.parse(appointmentList.get(index).getEndTime());
                long diffMs = startTime.getTime() - endTime.getTime();
                long diffSec = diffMs / 1000;
                long min = diffSec / 60;
                if (min < -10 && ((index + 1) < size) && appointmentList.get(index).getAppointmentType().equals("0")) {
                    Date newEndTime = simpleDateFormat.parse(appointmentList.get(index + 1).getStartTime());
                    long nxtAptMs = startTime.getTime() - newEndTime.getTime();
                    long nxtAptSec = nxtAptMs / 1000;
                    int nxtAptMin = (int)nxtAptSec / 60;

                    if (nxtAptMin <= 0) {
                        changeAppointmentStatus(appointmentList.get(index + 1), nxtAptMin);
                        sendNotification(appointmentList.get(index + 1).getRequesterDeviceToken(), min, true);
                    }
                }
            }
        }
    }
}

```

Figure 4.13. Schedule Manipulations due to Delay.

5. TESTING AND EVALUATION

Testing requires an Android mobile phone and register to this application as either a student or professor, then login. This application also takes care of authentication where a person will be able to login only if his credentials are valid. This can be evaluated based on the features provided and validations performed in the application. Each module of the application is tested with all the possible test cases.

5.1. Launching the Application

When the user launches the application for the first time he will see the login screen.

Figure 5.1 shows the launching screen and the login screen of the application.

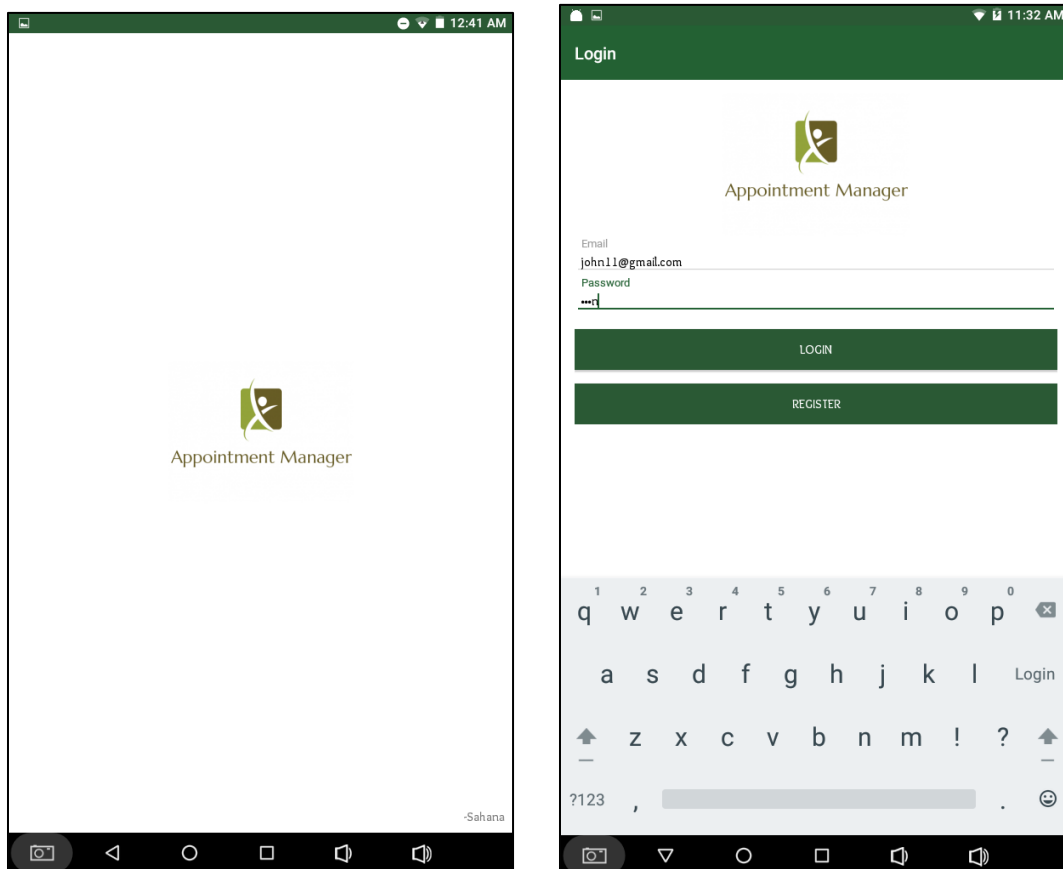


Figure 5.1. Launching Screen and Login Screen.

5.2. Positive and Negative Test Cases

Figure 5.2(a) shows the positive test case where a person could successfully login to her account while figure 5.2(b) shows a negative test case where application manages to take care of authentication. If the user enters wrong credentials or wrong email format, the application lets the user know by displaying a text below and prevents log in.

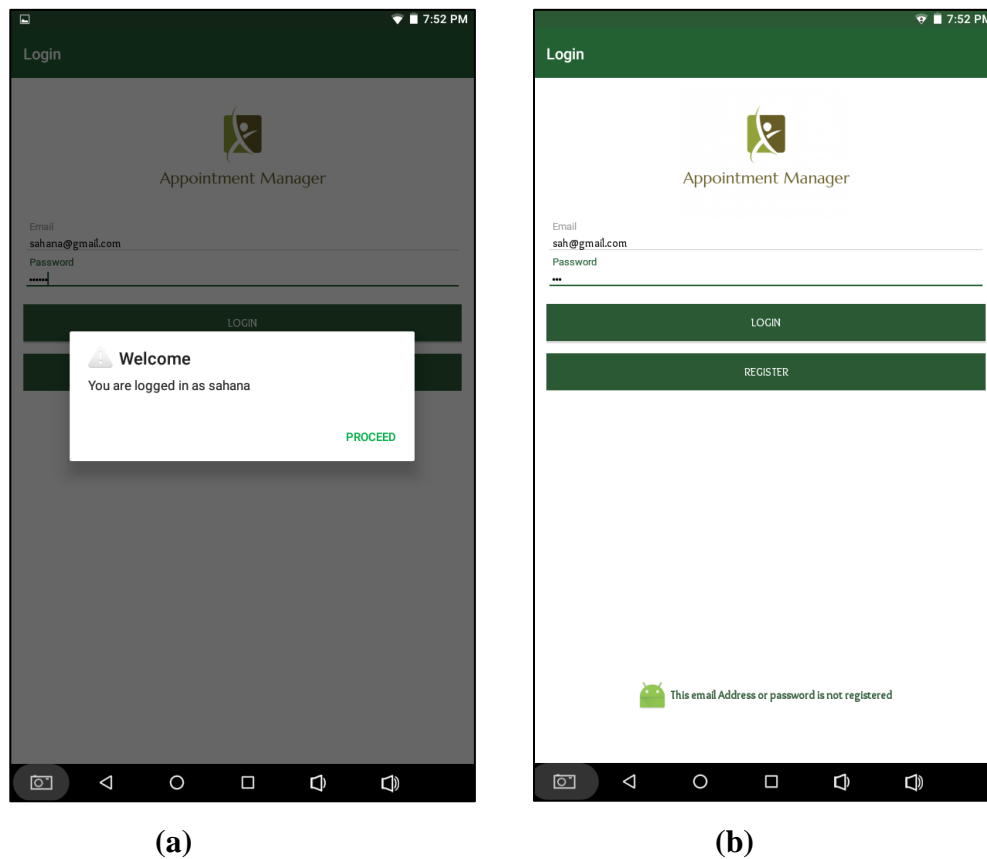


Figure 5.2. Successful and Not Successful Login Screens..

5.3. Requesting an Appointment Testing

For requesting an appointment, subject field cannot be empty as shown in figure 5.3(a). The professor will either approve or deny the request by seeing the subject entered by the student which shows the purpose of appointment. Figure 5.3(b) shows the requesting appointment is successful.

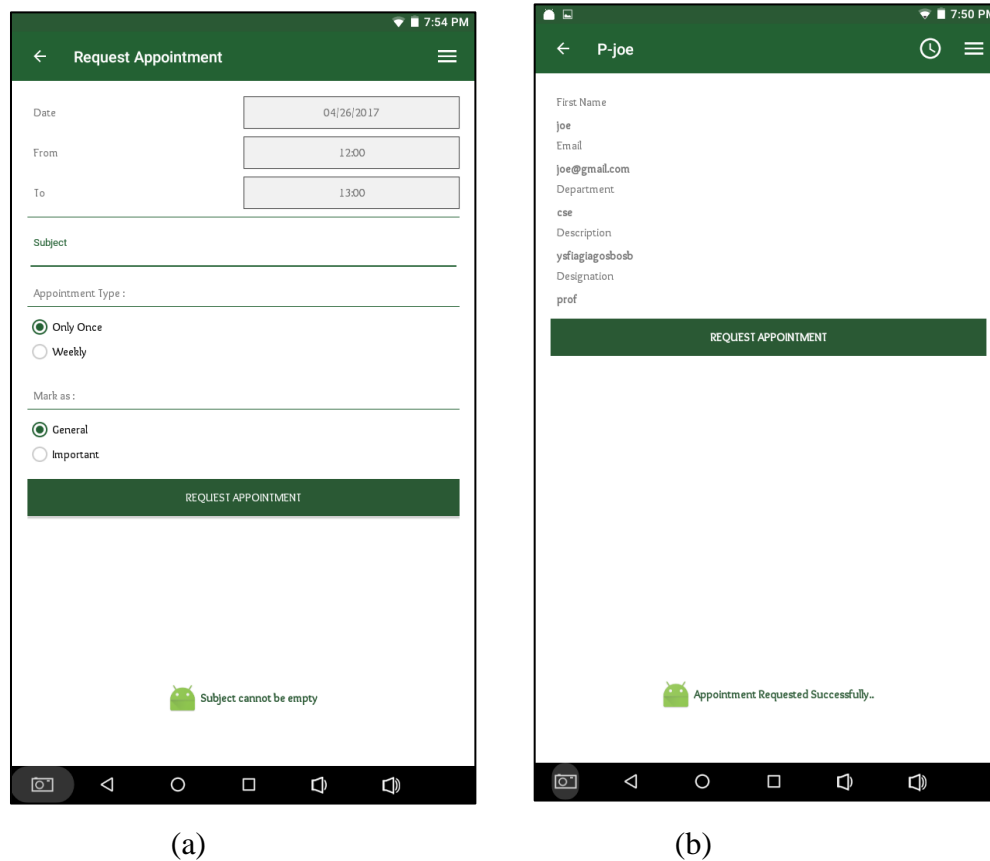


Figure 5.3. Requesting Appointment Test Case.

5.4. Request During Office/Free Hours

Figure 5.4(a) displays the scenario where a student is trying to request an appointment during non-office hours of the professor. But the application preventing the user to do so by prompting a message “Please check the professor’s office hours and request again” so that user goes back to check the office, free hours along with already booked times of the professor as shown in figure 5.4(b).

← Request Appointment

Date: 04/26/2017

From: 12:00

To: 12:15

Subject: Question about my grades

Appointment Type :

☒ Only Once

☐ Weekly

Mark as :

☒ General

☐ Important

REQUEST APPOINTMENT

Please check professor's office hours and request again.

(a)

← Schedule

Choose Date

04/28/2017

Booked Appointment

Free Hours

12:00-14:00

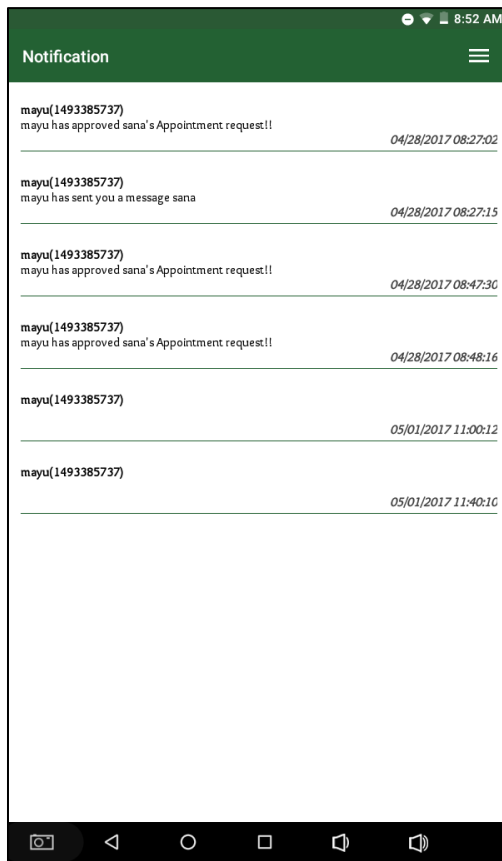
Office Hours

(b)

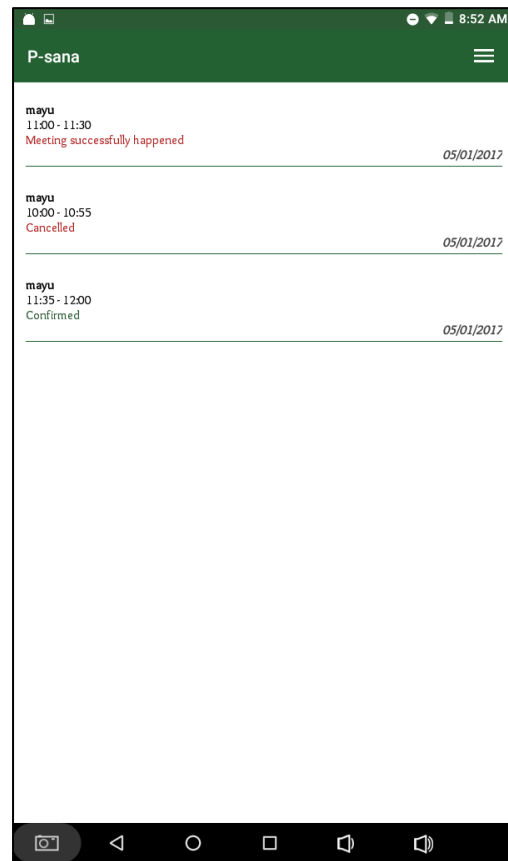
Figure 5.4. Requesting Appointment during Non-Office Hours.

5.5. Notifications

Figure 5.5(a) displays the notifications of the student indicating professor has approved request, cancelled request or sent some messages. Figure 5.5(b) displays the My Appointments menu of the student where it indicates the status of all his/her appointments.



(a)



(b)

Figure 5.5. Notifications and My Appointments Status.

5.6. Professor taking Leave

If a professor wanted to take a leave for one day or two, he can simply click a button in his profile to cancel all the scheduled appointments for the day and notifies the user about it and prompts them to request a new appointment again. Figure 5.6 shows the screen where a professor can click a button to declare leave.

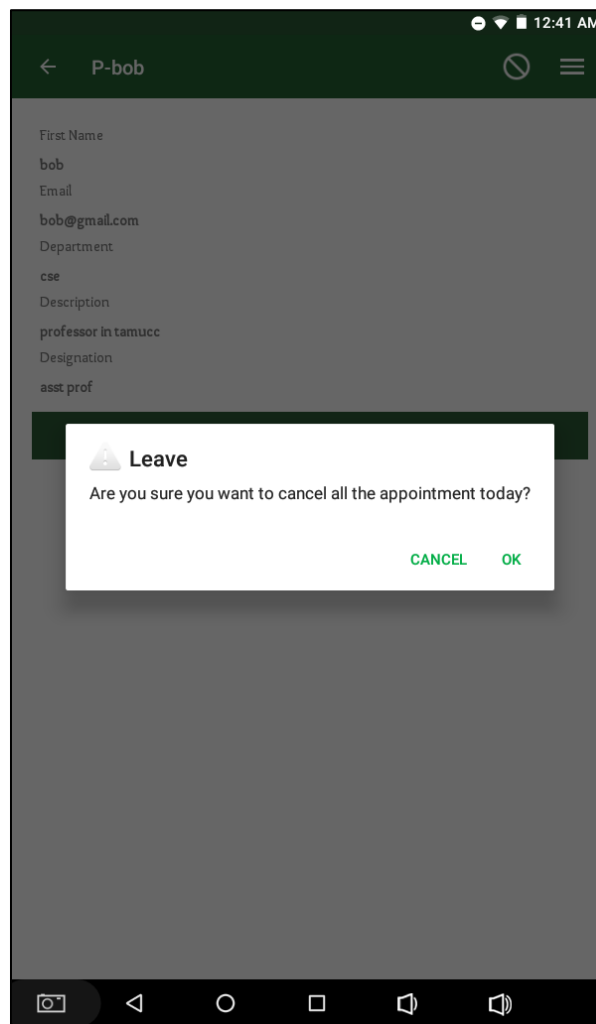


Figure 5.6. Professor taking Leave.

5.7. Student Starting the Appointment

Student manually clicks the start button before starting the appointment and clicks the done button once the appointment is done. Figure 5.7(a) shows both the start and cancel buttons. Student can click on cancel to cancel his appointment at any time. Once the student clicks the start button, cancel button will be changed to done button allowing the student to click it to at the end of the meeting.

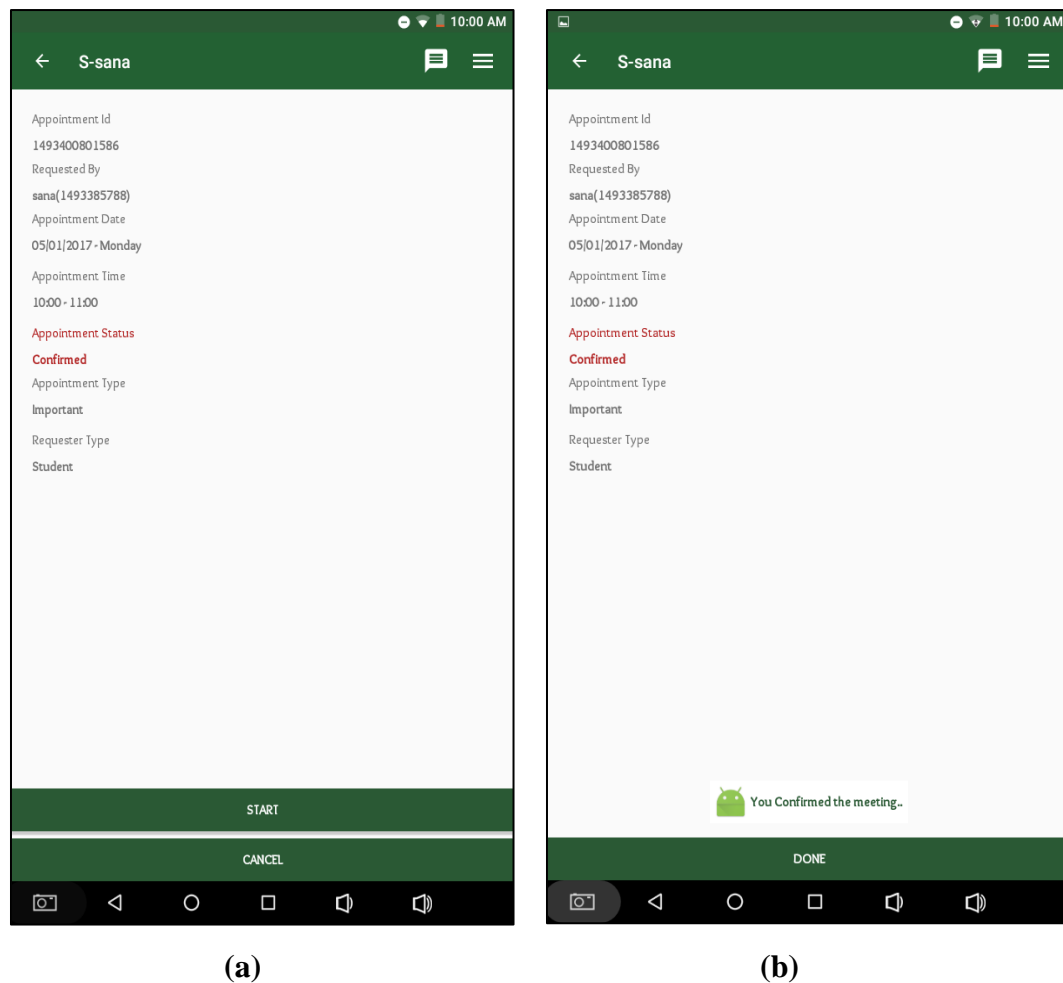


Figure 5.7. Student starting and ending Appointment.

5.8. Student Starting Negative Test Case

Student cannot start the appointment before the appointment scheduled date. If he attempts to start, the application prompts a message indicating “You can’t start the meeting before the date” and prevents starting as shown in figure 5.8.

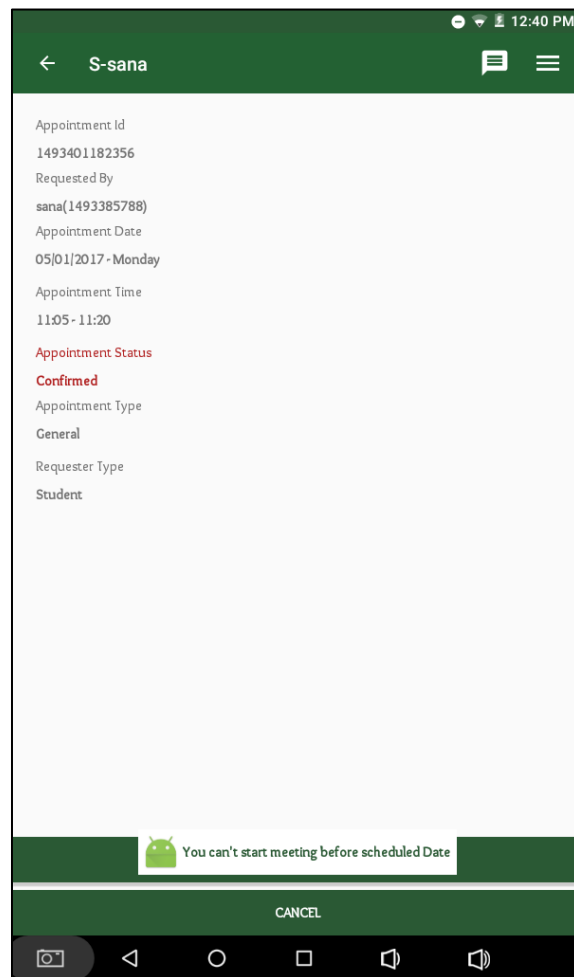


Figure 5.8. Student attempting to start before scheduled date .

5.9. Priorities

The already scheduled general appointment will be rescheduled if somebody requests an important appointment for the same time. Figure 5.9 displays how the status of appointments look like in professor's application.

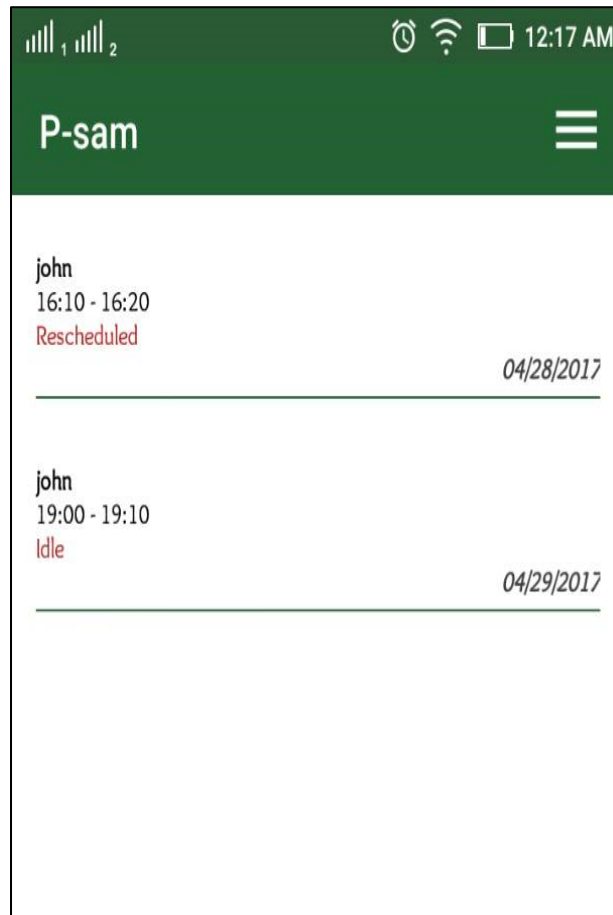


Figure 5.9. Handling priorities.

5.10. Rescheduling due to the Delay

As shown in figure 5.10 (a), initially 3 students scheduled appointments individually with professor. But as there was a delay in the first appointment, second appointment has rescheduled from to 9:45 am as shown in figure 5.10 (b). As the third appointment is not affected by the above delay, there is no change in it.

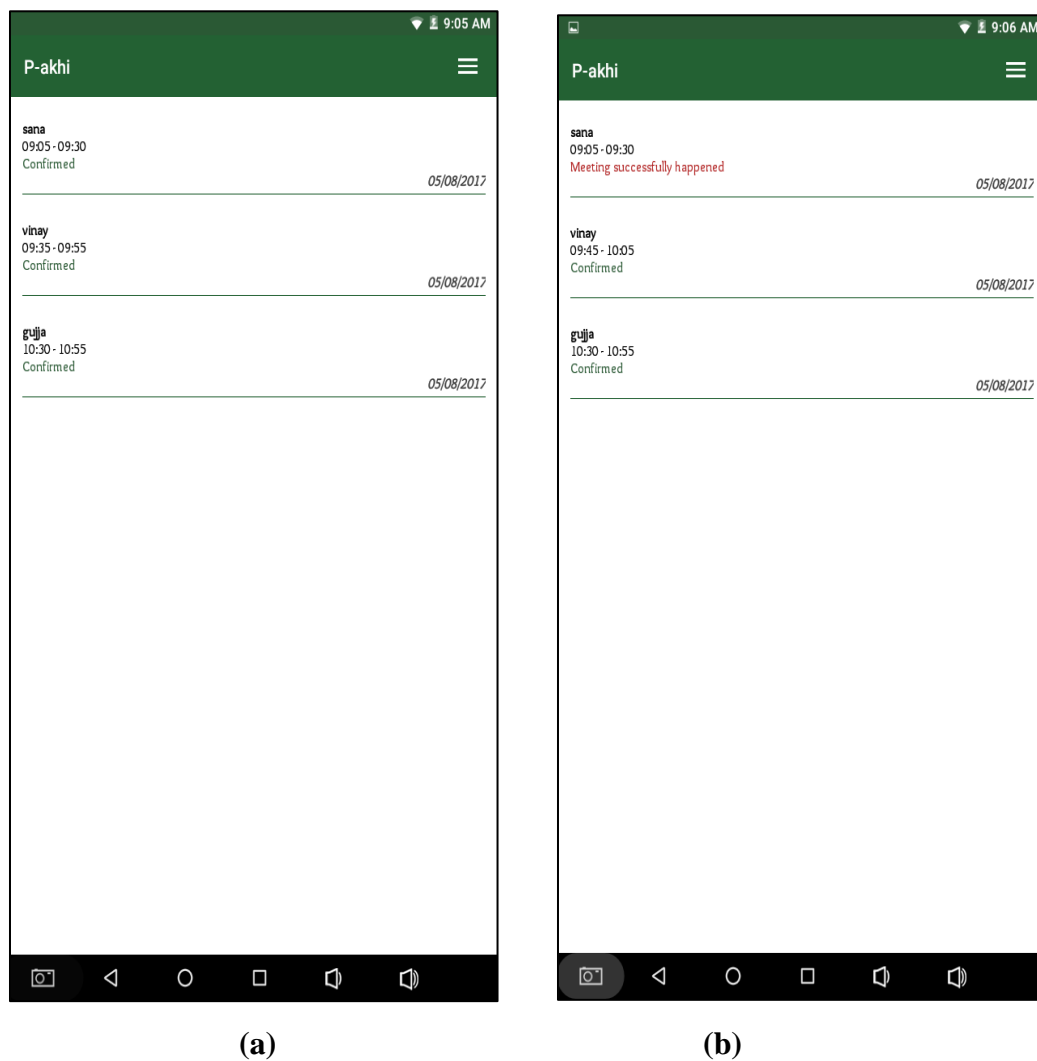


Figure 5.10. Rescheduling due to delay.

5.11. Delay Handling

The figure 5.11 (a) indicates that the first appointment has started at 9:05 am and as per figure 5.11(b), it ended at 9:45 am. But it was scheduled to end at 9:30 am as shown in figure 5.11 (b). Thus, results in 15-minute delay. As soon as the first appointment is done, the second person in the list will be notified about the delay as in figure 5.11 (c).

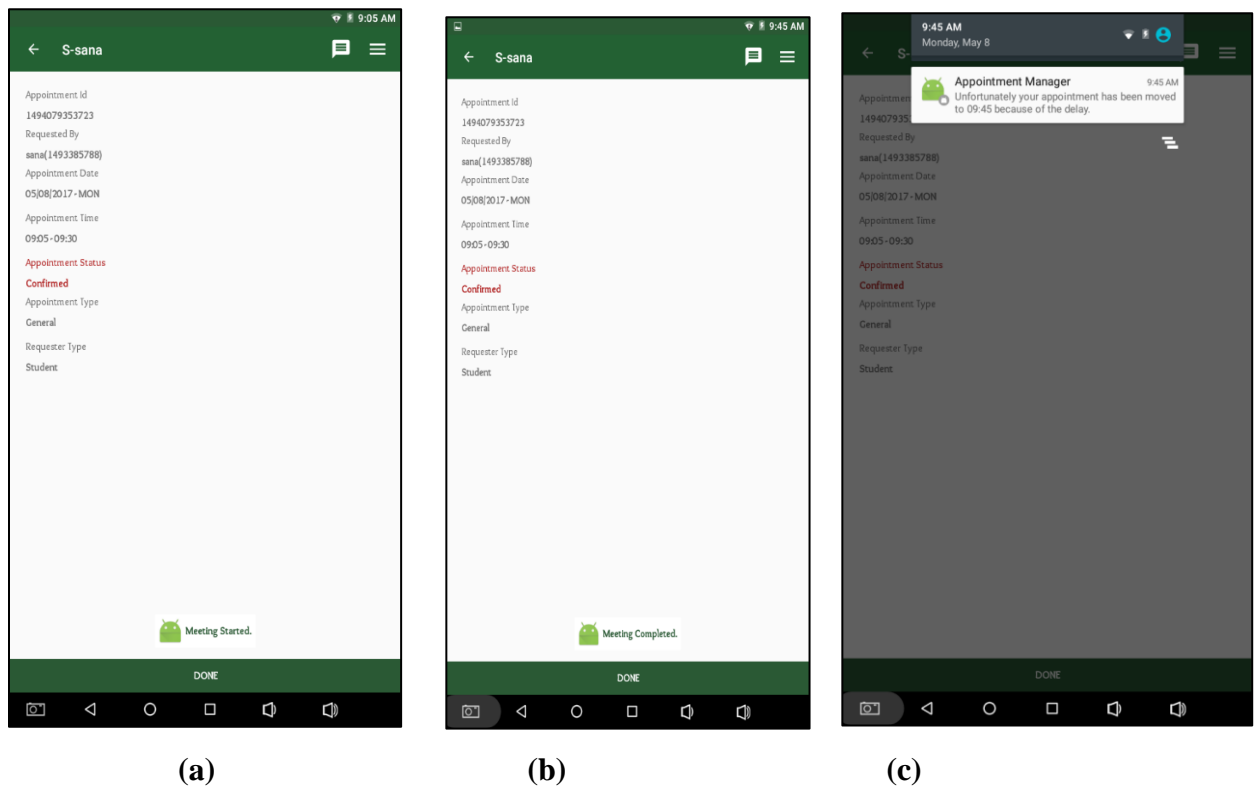


Figure 5.11. Delays handling.

5.12. Other Test Cases

Table 5.1: Test Results

TEST CASES	WORKING
Email accepts only valid type	YES
Valid username and password	YES
Empty fields are not accepted during registration/login	YES
Status (Confirmed/Idle/Rescheduled/Done) of appointments are properly updated	YES
Subject for requesting appointment should not be empty	YES
Start time for an appointment cannot be after end time.	YES
Appointments cannot be scheduled for days already passed.	YES
Completed appointments are removed form list of 'My Appointments'	YES
Office hours and Free hours' changes are properly updated for students' view	YES
Free hours will be removed at the end of the day	YES
Professor to Professor appointment will be set to high priority by default	YES
Message field cannot be empty for sending messages	YES
Appropriate notifications must be sent for individual users	YES
Student cannot start the meeting before scheduled date	YES
Notifications must be sent indicating appropriate delays	YES
Delays should push only those appointments that face the impact	YES

Appointment will be rescheduled to the next available time either the same day or the next day.	YES
Appointments will be cancelled automatically only if it cannot be rescheduled for the next day.	YES
Appointments can be cancelled by the professor	YES
Appointments can be cancelled by the student.	YES
Important appointments cannot be rescheduled (Explained below)	Partially*
General appointments will be rescheduled only if the same slot is requested by a high priority appointment.	YES
New High priority appointment requests cannot trigger rescheduling of general appointments that are waiting to happen in less than 24 hours.	YES
Professor's 'LEAVE' feature cancels all appointments for the day	YES
Professor can toggle his appointments as High Priority or General	YES
Users account remains intact with same credentials until logged out	YES

All test cases mentioned are working fine except the one which has been marked as 'partially' in table 5.1. If an important appointment is scheduled at the end of the day, it might be rescheduled due to the prior appointment's delay. This case could not be handled by the application as it is using an array list and handling all the appointments one by one and notifying people about the possible changes. The recommendation to all the users of this application is not to schedule an important appointment at the end of office hours every day as it might get rescheduled due to previous appointment's delay.

5.13. User's Evaluation

The results obtained after the application was evaluated by few users are shown in table 5.2.

Table 5.2: User Ratings

User's Ratings	Result (On a scale of 1-10) *Rounded off to nearest Integer
Login Authentication	10
Appointment Scheduling	9
Notifications	9
Messaging Service	10
Delay Handling	7
Prioritized Handling	8
User Interface	8
Professor-Professors Appointments	9
Application Performance	9
Overall Satisfaction	8

5. CONCLUSION AND FUTURE WORK

The application design was expected to provide a time saving service to its users by providing an android platform. The application should be able to solve the waiting process of the users by implementing an interface to request, schedule and manage appointments. Students should be able to view the profiles of the Professors where their office times, free times and already booked times are mentioned. This app helps professors to save their time instead of wasting time in manually scheduling individual appointments. It also helps students to know the updates regarding his appointment time to time.

Features that are to be implemented in future are as follows:

- Scheduling meeting for more than two professors by integrating their schedules.
- Time based reminders.
- Taking a set of student's available times so that application can reschedule accordingly if needed.
- Can be implemented for physicians or other companies.

BIBLIOGRAPHY and REFERENCES:

1. Android Studio https://en.wikipedia.org/wiki/Android_Studio
2. Appointment scheduling application <https://www.appointment-plus.com/>
3. Chongjunand Yan, Jiafu Tang, “Sequential Appointment Scheduling Problem with General Patient Choice” Proceeding of the 11th World Congress on Intelligent Control and Automation, China, 2014.
4. Paulien Koeleman, Ger Koole, “Appointment scheduling using optimization via simulation” Proceedings of the 2012 Winter Simulation Conference, Netherlands, 2012.
5. Firebase Cloud Messaging <https://firebase.google.com/docs/cloud-messaging/>
6. CASA Writing center TAMUCC <http://casa.tamucc.edu/wc.php>
7. Thomas A. Frasher, Todd M. Fitch, Steven A. Sholtis, “Method and system for priority-based appointment scheduling”, 2011.

APPENDIX A: CODE SNIPPETS

This sections contains the code snippets of the main features of the project.

LoginActivity.java

```

@Override
protected int getLayoutResourceId() {
    return R.layout.activity_login;
}

@Override
public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);

    if (Constant.FIREBASE_TOKEN == null)
        Constant.FIREBASE_TOKEN = FirebaseInstanceId.getInstance().getToken();

    ButterKnife.bind(this);
    customFont = TypefaceUtils.getRegularFont(this);

    @OnClick({R.id.btnLogin, R.id.btnRegister, R.id.btnUpdateLocation,
R.id.btnViewLocation})
    public void onClick(View view) {
        switch (view.getId()) {
            case R.id.btnUpdateLocation:

                Intent mIntent = new Intent(LoginActivity.this, UpdateLocation.class);
                mIntent.putExtra("EXTRA_IS_UPDATE", true);
                startActivity(mIntent);

                break;
            case R.id.btnViewLocation:

                Intent viewIntent = new Intent(LoginActivity.this, UpdateLocation.class);
                viewIntent.putExtra("EXTRA_IS_UPDATE", false);
                startActivity(viewIntent);

                break;
            case R.id.btnLogin:

                if (Utils.isEmailValid(edtEmail)) {
                    if (edtPassword.getText().toString().length() == 0)

```

```

        Utils.displayToast(getString(R.string.error_password_empty), this);
    else
        doLogin(edtEmail.getText().toString(), edtPassword.getText().toString());
    } else Utils.displayToast(getString(R.string.error_invalid_email), this);
    break;
case R.id.btnRegister:
    startActivity(new Intent(LoginActivity.this, Registration.class));
    break;
}
}

private void doLogin(final String email, final String password) {

    try {
        showProgress();

        FirebaseDatabase database = FirebaseDatabase.getInstance();
        final DatabaseReference userRef =
        database.getReference(Constant.TABLE_USER);

        userRef.addListenerForSingleValueEvent(new ValueEventListener() {
            @Override
            public void onDataChange(DataSnapshot dataSnapshot) {

                boolean isAvailable = false;
                stopProgress();

                for (DataSnapshot snapshot : dataSnapshot.getChildren()) {
                    DataUser obj = snapshot.getValue(DataUser.class);
                    if (obj.getUserEmail().equals(email) &&
                    obj.getUserPassword().equals(password)) {

                        final SharedPreferences prefs = new ObscuredSharedPreferences(
                            LoginActivity.this,
                            LoginActivity.this.getSharedPreferences(Constant.PREFERENCES,
                                Context.MODE_PRIVATE));

                        prefs.edit().putString(Constant.USER_EMAIL,
                            obj.getUserEmail()).apply();
                        prefs.edit().putString(Constant.USER_ID, obj.getId()).apply();
                        prefs.edit().putString(Constant.USER_NAME,
                            obj.getUserName()).apply();
                        prefs.edit().putString(Constant.USER_TYPE,
                            obj.getUserType()).apply();
                        showSucessDialog(obj);
                        return;
                    }
                }
            }
        });
    }
}

```

```

        }
    }

    if (!isAvailable)
        Utils.displayToast(getString(R.string.error_invalid_email_or_password),
        LoginActivity.this);
    }

    @Override
    public void onCancelled(DatabaseError databaseError) {
        stopProgress();
        Utils.displayToast(getString(R.string.something_went_wrong),
        LoginActivity.this);
    }
    });
} catch (Exception ex) {
    ex.printStackTrace();
}
}

```

ProfessorHomeActivity.java

```

@Override
protected int getLayoutResourceId() {
    return R.layout.act_student_home;
}

private void initUi() {

    SharedPreferences prefs = new ObscuredSharedPreferences(
        ProfessorHomeActivity.this,
        ProfessorHomeActivity.this.getSharedPreferences(Constant.PREFERENCES,
        Context.MODE_PRIVATE));
    isProfessor = prefs.getString(Constant.USER_TYPE,
    "").equals(Constant.PROFESSOR);

    isMyAppointment = getIntent().getBooleanExtra("IS_MY_APPOINTMENT",
    false);
    myUserId = prefs.getString(Constant.USER_ID, "");

    if (Constant.FIREBASE_TOKEN == null)
        Constant.FIREBASE_TOKEN = FirebaseInstanceId.getInstance().getToken();

    setSupportActionBar(toolbar);
}

```



```

        getSupportActionBar().setTitle(isProfessor ? "P-" +
        prefs.getString(Constant.USER_NAME, "") : "S-" +
        prefs.getString(Constant.USER_NAME, ""));

        // getSupportActionBar().setDisplayHomeAsUpEnabled(true);

        LinearLayoutManager llm = new LinearLayoutManager(this);
        llm.setOrientation(LinearLayoutManager.VERTICAL);
        recyclerProfessorList.setLayoutManager(llm);
        getAppointmentList();
    }
    private void getAppointmentList() {

        // 0 - idle
        // 1 - Confirmed
        // 2 - cancelled
        // 3 - rescheduled
        // 4 - meeting successfully happened
        // 5 - started

        try {
            showProgress();

            FirebaseDatabase database = FirebaseDatabase.getInstance();
            final DatabaseReference profRef =
            database.getReference(Constant.TABLE_APPOINTMENT);

            profRef.addListenerForSingleValueEvent(new ValueEventListener() {

                @Override
                public void onDataChange(DataSnapshot dataSnapshot) {

                    SimpleDateFormat simpleDateFormat = new
                    SimpleDateFormat("MM/dd/yyyy", Locale.US);
                    String todayDate =
                    simpleDateFormat.format(Calendar.getInstance().getTime());

                    stopProgress();
                    try {

                        for (DataSnapshot snapshot : dataSnapshot.getChildren()) {
                            DataAppointment obj = snapshot.getValue(DataAppointment.class);

                            if (isMyAppointment) {
                                if (obj.getRequesterId().equals(myUserId)) {
                                    appointmentList.add(obj);
                                }
                            }
                        }
                    }
                }
            });
        }
    }

```

```

    }
    } else if (isProfessor) {
        if (obj.getResponderId().equals(myUserId)) {

            Date date1 = simpleDateFormat.parse(todayDate);
            Date date2 = simpleDateFormat.parse(obj.getAppointmentDate());

            if (date2.compareTo(date1) >= 0 &&
!obj.getAppointmentStatus().equals("2"))
                appointmentList.add(obj);
        }
    }
}

if (appointmentList.size() == 0)
    showSnackBar(parentPanel, getString(R.string.no_appts_found));
else {
    adapter = new AppointmentListAdapter(appointmentList,
ProfessorHomeActivity.this, isMyAppointment);
    recyclerProfessorList.setAdapter(adapter);
}

} catch (Exception ex) {
    ex.printStackTrace();
}
}

@Override
public void onCancelled(DatabaseError databaseError) {
    stopProgress();
    Utils.displayToast(getString(R.string.something_went_wrong),
ProfessorHomeActivity.this);
}
});
} catch (Exception ex) {
    ex.printStackTrace();
    Utils.displayToast(getString(R.string.something_went_wrong),
ProfessorHomeActivity.this);
}
}
}

```

BookAppointment.java

```

private void bookAppointment(String appointmentDate, String timestamp) {

```

```

Utils.debug("appointmentDate : " + appointmentDate + " / " + timestamp);

// Appointment type

// 0 - General
// 1 - Important

try {
    showProgress();
    String id = String.valueOf(System.currentTimeMillis());
    Utils.debug("Appointment id generation : " + id);

    final DataAppointment dataAppointment = new DataAppointment();
    dataAppointment.setAppointmentDate(appointmentDate);

    Calendar c = Calendar.getInstance();
    c.setTime(new SimpleDateFormat("MM/dd/yyyy",
Locale.US).parse(appointmentDate));
    int dayOfWeek = c.get(Calendar.DAY_OF_WEEK);

    dataAppointment.setAppointmentDay(String.valueOf(dayOfWeek));
    dataAppointment.setAppointmentType(radGeneral.isChecked() ? "0" : "1");
    dataAppointment.setApptWeight(radGeneral.isChecked() ? "0" : "2");
    dataAppointment.setStartTime(txtFromTime.getText().toString());
    dataAppointment.setEndTime(txtToTime.getText().toString());
    dataAppointment.setApptId(id);
    dataAppointment.setRequesterDeviceToken(Constant.FIREBASE_TOKEN);

    dataAppointment.setRequestedTime(timestamp);
    dataAppointment.setRequesterId(prefs.getString(Constant.USER_ID, ""));
    dataAppointment.setRequesterName(prefs.getString(Constant.USER_NAME,
    ""));
    dataAppointment.setResponderId(dataProfessor.getProfessorId());
    dataAppointment.setResponderName(dataProfessor.getProfessorName());
    dataAppointment.setAppointmentStatus("0");
    dataAppointment.setRequesterType(prefs.getString(Constant.USER_TYPE, ""));

    final FirebaseDatabase database = FirebaseDatabase.getInstance();
    final DatabaseReference apptRef =
database.getReference(Constant.TABLE_APPOINTMENT);
    apptRef.child(id).setValue(dataAppointment);

    apptRef.addListenerForSingleValueEvent(new ValueEventListener() {
        @Override
        public void onDataChange(DataSnapshot dataSnapshot) {

```

```

        if (!isNotificationSent) {
            String message = dataAppointment.getRequesterName()
                + " has requested an appointment request to " +
dataAppointment.getResponderName();
            new SendNotification(Constant.PROFESSOR,
                dataProfessor.getProfessorDeviceTkn(), message);
            saveNotificationToFirebase(dataAppointment, message);
        }
        stopProgress();
        Utils.displayToast(getString(R.string.appointment_requested),
BookAppointment.this);
        finish();
    }

    @Override
    public void onCancelled(DatabaseError databaseError) {
        stopProgress();
        Utils.displayToast(getString(R.string.something_went_wrong),
BookAppointment.this);
    }
});

} catch (Exception ex) {
    stopProgress();
    Utils.displayToast(getString(R.string.something_went_wrong),
BookAppointment.this);
    ex.printStackTrace();
}
}

private void saveNotificationToFirebase(DataAppointment newData, String message)
{

    try {
        showProgress();

        final FirebaseDatabase database = FirebaseDatabase.getInstance();
        final DatabaseReference appRef =
database.getReference(Constant.TABLE_NOTIFICATIONS);

        DataNotification dataNotification = new DataNotification();
        String id = String.valueOf(System.currentTimeMillis());

        dataNotification.setNotificationId(id);
        dataNotification.setNotificationMsg(message);
    }
}

```

```

dataNotification.setReceiverId(newData.getResponderId());
dataNotification.setReceiverName(newData.getRequesterName());

dataNotification.setSenderId(newData.getRequesterId());
dataNotification.setSenderName(newData.getResponderName());
dataNotification.setTimestamp(new SimpleDateFormat("MM/dd/yyyy
hh:mm:ss", Locale.US).format(Calendar.getInstance().getTime()));

apptRef.child(id).setValue(dataNotification);
apptRef.addListenerForSingleValueEvent(new ValueEventListener() {

    @Override
    public void onDataChange(DataSnapshot dataSnapshot) {
        stopProgress();
    }

    @Override
    public void onCancelled(DatabaseError databaseError) {
        stopProgress();
        Utils.displayToast(getString(R.string.something_went_wrong),
BookAppointment.this);
    }
});

} catch (Exception ex) {
    stopProgress();
    Utils.displayToast(getString(R.string.something_went_wrong),
BookAppointment.this);
    ex.printStackTrace();
}
}

private void recursiveBooking() {

    try {
        DateFormat dateFormat = new SimpleDateFormat("MM/dd/yyyy HH:mm:ss",
Locale.US);
        Calendar c = Calendar.getInstance();
        String timestamp = dateFormat.format(c.getTime());

        c.setTime(new SimpleDateFormat("MM/dd/yyyy",
Locale.US).parse(txtDate.getText().toString()));
        int dayOfWeek = c.get(Calendar.DAY_OF_WEEK);
        Utils.debug("dayOfWeek : " + dayOfWeek +
        new SimpleDateFormat("MM/dd/yyyy",

```

```

Locale.US).parse(txtDate.getText().toString()))
    ;

    for (int i = 1; i <= 12; i++) {

        Utils.debug("i * 7 : " + i * 7);
        c.add(Calendar.DATE, 7);

        int date = c.get(Calendar.DATE);
        int month = c.get(Calendar.MONTH);
        int year = c.get(Calendar.YEAR);

        StringBuilder sb = new StringBuilder();
        sb.append(month + 1).append("/").append(date).append("/").append(year);
        Utils.debug("sb.toString : " + sb.toString());
        bookAppointment(sb.toString(), timestamp);
    }

    finish();
} catch (Exception ex) {
    ex.printStackTrace();
}
}

@Override
public void onTimeSet(RadialPickerLayout view, int hourOfDay, int minute, int
hourOfDayEnd, int minuteEnd) {

    String hourString = hourOfDay < 10 ? "0" + hourOfDay : "" + hourOfDay;
    String minuteString = minute < 10 ? "0" + minute : "" + minute;
    String hourStringEnd = hourOfDayEnd < 10 ? "0" + hourOfDayEnd : "" +
hourOfDayEnd;
    String minuteStringEnd = minuteEnd < 10 ? "0" + minuteEnd : "" + minuteEnd;
    String time = "You picked the following time: From - " + hourString + "h" +
minuteString + " To - " + hourStringEnd + "h" + minuteStringEnd;

    StringBuilder from = new StringBuilder();
    from.append(hourString).append(":").append(minuteString);

    StringBuilder to = new StringBuilder();
    to.append(hourStringEnd).append(":").append(minuteStringEnd);
    txtFromTime.setText(from.toString());
    txtToTime.setText(to.toString());
    Utils.debug(time);
}

```

Rescheduling.java

```

public class Reshceduler {

    private static final String APPT_TYPE_IMPORTANT = "1";

    private Context context;
    private List<DataAppointment> appointmentList = new ArrayList<>();
    private String fromTime = "", toTime = "", selectedDate = "";
    private ArrayList<String> todayBusyHours = new ArrayList<>();
    private ArrayList<String> todayFreeHours = new ArrayList<>();
    private boolean isRecursive = false, isGeneral = false, isNotificationSent = false;
    private SharedPreferences prefs;
    private DataProfessor dataProfessor;

    public Reshceduler(Context con
        , List<DataAppointment> appointmentList
        , String fromTime
        , String toTime
        , String selectedDate
        , boolean isRecursive
        , boolean isGeneral
        , DataProfessor dataProfessor
        , ArrayList<String> todayBusyHours
        , ArrayList<String> todayFreeHours) {

        Utils.debug("Inside Resheduler...");

        this.context = con;
        this.appointmentList = appointmentList;
        this.fromTime = fromTime.replace(":", ".");
        this.toTime = toTime.replace(":", ".");
        this.isRecursive = isRecursive;
        this.selectedDate = selectedDate;
        this.isGeneral = isGeneral;
        this.dataProfessor = dataProfessor;

        prefs = new ObscuredSharedPreferences(
            con, con.getSharedPreferences(Constant.PREFERENCES,
            Context.MODE_PRIVATE));

        if (checkProfessorFreeHours(todayBusyHours, todayFreeHours, fromTime,
            toTime)) {
            if (checkFreeTimeAvailable()) {
                bookAppointment();
            } else {
                Utils.displayToast("Unable to Reschedule. Please try different date.",

```

```

context);
    }
    } else {
        Utils.displayToast("Professor doesn't have any schedule today. Please try
different date.", context);
        return;
    }
}

```

```

private boolean checkFreeTimeAvailable() {

    boolean canBookAppointment = false;
    float requestedStartTime = Float.parseFloat(fromTime);
    float requestedEndTime = Float.parseFloat(toTime);

    Utils.debug("checkFreeTimeAvailable....");

    int size = appointmentList.size();

    Utils.debug("checkFreeTimeAvailable appointmentList.size() : " +
appointmentList.size());

    if (size == 0) {
        return true;
    }

    for (int index = 0; index < size; index++) {

        Utils.debug("Inside for loop.. : " + appointmentList.size());

        float existingStartTime =
Float.parseFloat(appointmentList.get(index).getStartTime().replace(":", "."));
        float existingEndTime =
Float.parseFloat(appointmentList.get(index).getEndTime().replace(":", "."));

        if ((requestedStartTime <= existingStartTime && requestedEndTime >=
existingStartTime) ||
            (requestedStartTime <= existingEndTime && requestedEndTime >=
existingEndTime)) {

            if
(appointmentList.get(index).getAppointmentType().equals(APPT_TYPE_IMPORTANT)
) {

```



```

        return canBookAppointment;
    } else
        cancelAndReschedule(index);
    } else
        canBookAppointment = true;
    }
    return canBookAppointment;
}

private boolean checkProfessorFreeHours(ArrayList<String> officeHours,
                                         ArrayList<String> freeHours, String fromTime, String
toTime) {

    boolean isFreeHoursAvailable = false;
    float requestedStartTime = Float.parseFloat(fromTime.replace(":", "."));
    float requestedEndTime = Float.parseFloat(toTime.replace(":", "."));

    if (freeHours != null && freeHours.size() == 0 && officeHours != null &&
officeHours.size() == 0) {
        Utils.displayToast("No time available on this day..", context);
        return isFreeHoursAvailable;
    }

    int size = officeHours.size();

    for (int index = 0; index < size; index++) {

        Utils.debug("Inside for loop free hours....");

        String string = officeHours.get(index);
        String[] parts = string.split("-");

        float existingStartTime = Float.parseFloat(parts[0].replace(":", "."));
        float existingEndTime = Float.parseFloat(parts[1].replace(":", "."));

        Utils.debug("Time " + requestedStartTime + " // " + requestedEndTime + " // "
+
        existingStartTime + " // " + existingEndTime);

        if ((requestedStartTime >= existingStartTime && requestedEndTime <=
existingEndTime)) {

            Utils.debug("isFreeHoursAvailable.. " + isFreeHoursAvailable);

            isFreeHoursAvailable = true;

```

```

        return isFreeHoursAvailable;
    } else
        isFreeHoursAvailable = false;
    }

    int freeHoursSize = freeHours.size();

    for (int index = 0; index < freeHoursSize; index++) {

        Utils.debug("Inside for loop freeHours hours....");

        String string = freeHours.get(index);
        String[] parts = string.split("-");

        float existingStartTime = Float.parseFloat(parts[0].replace(":", "."));
        float existingEndTime = Float.parseFloat(parts[1].replace(":", "."));

        Utils.debug("freeHours " + requestedStartTime + " // " + requestedEndTime +
            " // " +
            existingStartTime + " // " + existingEndTime);

        if ((requestedStartTime >= existingStartTime && requestedEndTime <=
            existingEndTime)) {
            isFreeHoursAvailable = true;
            return isFreeHoursAvailable;
        } else
            isFreeHoursAvailable = false;
        }
    return isFreeHoursAvailable;
}

private boolean cancelAndReschedule(int cancelIndex) {

    Utils.debug("Cancel and Reschedule..");

    cancelExistingAppointment(appointmentList.get(cancelIndex));

    boolean canBookAppointment = false;
    float requestedStartTime = Float.parseFloat(fromTime);
    float requestedEndTime = Float.parseFloat(toTime);

    int size = appointmentList.size();

    for (int index = 0; index < size; index++) {

```

```

        float existingStartTime =
Float.parseFloat(appointmentList.get(index).getStartTime().replace(":", "."));
        float existingEndTime =
Float.parseFloat(appointmentList.get(index).getEndTime().replace(":", "."));

        if ((requestedStartTime <= existingStartTime && requestedEndTime >=
existingStartTime) ||
            (requestedStartTime <= existingEndTime && requestedEndTime >=
existingEndTime)) {

            if
(appointmentList.get(index).getAppointmentType().equals(APPT_TYPE_IMPORTANT)
) {

                }

            } else
                canBookAppointment = true;
        }
        return canBookAppointment;
    }

    private void bookAppointment() {
        if (!isRecursive) {
            DateFormat dateFormat = new SimpleDateFormat("MM/dd/yyyy HH:mm:ss",
Locale.US);
            Calendar c = Calendar.getInstance();
            String timestamp = dateFormat.format(c.getTime());
            bookAppointment(selectedDate, timestamp);
        }
    }

    private void bookAppointment(String appointmentDate, String timestamp) {

        Utils.debug("appointmentDate : " + appointmentDate + " / " + timestamp);

        // Appointment type

        // 0 - Regular
        // 1 - Urgent

        try {

            String id = String.valueOf(System.currentTimeMillis());

```

```

Utils.debug("Appointment id generation : " + id);

final DataAppointment dataAppointment = new DataAppointment();
dataAppointment.setAppointmentDate(appointmentDate);

Calendar c = Calendar.getInstance();
SimpleDateFormat simpleDateFormat = new
SimpleDateFormat("MM/dd/yyyy", Locale.US);

c.setTime(simpleDateFormat.parse(appointmentDate));
int dayOfWeek = c.get(Calendar.DAY_OF_WEEK);

dataAppointment.setAppointmentDay(String.valueOf(dayOfWeek));
dataAppointment.setAppointmentType(isGeneral ? "0" : "1");
dataAppointment.setStartTime(fromTime.replace(":", ":"));
dataAppointment.setEndTime(toTime.replace(":", ":"));
dataAppointment.setApptId(id);
dataAppointment.setRequesterDeviceToken(Constant.FIREBASE_TOKEN);

System.out.println(timestamp);
System.out.println(Constant.FIREBASE_TOKEN);

dataAppointment.setRequestedTime(timestamp);
dataAppointment.setRequesterId(prefs.getString(Constant.USER_ID, ""));
dataAppointment.setRequesterName(prefs.getString(Constant.USER_NAME,
""));
dataAppointment.setResponderId(dataProfessor.getProfessorId());
dataAppointment.setResponderName(dataProfessor.getProfessorName());
dataAppointment.setAppointmentStatus("0");
dataAppointment.setRequesterType(prefs.getString(Constant.USER_TYPE, ""));
dataAppointment.setApptWeight(isGeneral ? "0" : "2");

final FirebaseDatabase database = FirebaseDatabase.getInstance();
final DatabaseReference apptRef =
database.getReference(Constant.TABLE_APPOINTMENT);
apptRef.child(id).setValue(dataAppointment);
apptRef.addListenerForSingleValueEvent(new ValueEventListener() {
    @Override
    public void onDataChange(DataSnapshot dataSnapshot) {

        if (!isNotificationSent) {

            String message = dataAppointment.getRequesterName()
                + " has requested an appointment request to " +
dataAppointment.getResponderName();

```

```

        new SendNotification(Constant.PROFESSOR,
            dataProfessor.getProfessorDeviceTkn(), message);
        // saveNotificationToFirebase(dataAppointment, message);
    }

    Utils.displayToast(context.getString(R.string.appointment_requested),
context);
    ((BookAppointment) context).finish();
    }

    @Override
    public void onCancelled(DatabaseError databaseError) {

        Utils.displayToast(context.getString(R.string.something_went_wrong),
context);
    }
    });

    } catch (Exception ex) {

        Utils.displayToast(context.getString(R.string.something_went_wrong), context);
        ex.printStackTrace();
    }
    }

    private void cancelExistingAppointment(DataAppointment d) {

        Utils.debug("cancelExistingAppointment...");

        try {

            ArrayList<String> officeHours = new ArrayList<>();
            ArrayList<String> freeHours = new ArrayList<>();

            try {
                Calendar cal = Calendar.getInstance();
                SimpleDateFormat sdf = new SimpleDateFormat("MM/dd/yyyy",
Locale.US);
                cal.setTime(sdf.parse(selectedDate));
                cal.add(Calendar.DAY_OF_MONTH, 1);

                officeHours = getBusyHours(getKey(cal.get(Calendar.DAY_OF_WEEK)));
                freeHours = getFreeHours(getKey(cal.get(Calendar.DAY_OF_WEEK)));
            }
        }
    }

```

```

String newDate = sdf.format(cal.getTime());
Utils.debug("New Date : " + newDate);

    if (checkProfessorFreeHours(officeHours, freeHours, d.getStartTIme(),
d.getEndTIme())) {
        String id = String.valueOf(System.currentTimeMillis());
        Utils.debug("Appointment id generation : " + id);

        final DataAppointment dataAppointment = new DataAppointment();
        dataAppointment.setAppointmentDate(newDate);

        Calendar c = Calendar.getInstance();
        c.setTime(new SimpleDateFormat("MM/dd/yyyy",
Locale.US).parse(newDate));
        int dayOfWeek = c.get(Calendar.DAY_OF_WEEK);

        dataAppointment.setAppointmentDay(String.valueOf(dayOfWeek));
        dataAppointment.setAppointmentType(isGeneral ? "0" : "1");
        dataAppointment.setStartTime(fromTime.replace(".", ":"));
        dataAppointment.setEndTime(toTime.replace(".", ":"));
        dataAppointment.setApptId(id);

dataAppointment.setRequesterDeviceToken(Constant.FIREBASE_TOKEN);

        dataAppointment.setRequestedTime(sdf.format(c.getTime()));
        dataAppointment.setRequesterId(prefs.getString(Constant.USER_ID, ""));

dataAppointment.setRequesterName(prefs.getString(Constant.USER_NAME, ""));
        dataAppointment.setResponderId(dataProfessor.getProfessorId());
        dataAppointment.setResponderName(dataProfessor.getProfessorName());
        dataAppointment.setAppointmentStatus("3");
        dataAppointment.setRequesterType(prefs.getString(Constant.USER_TYPE,
""));

        dataAppointment.setApptWeight("2");

        final FirebaseDatabase database = FirebaseDatabase.getInstance();
        final DatabaseReference apptRef =
database.getReference(Constant.TABLE_APPOINTMENT);
        apptRef.child(id).setValue(dataAppointment);
        apptRef.addListenerForSingleValueEvent(new ValueEventListener() {
            @Override
            public void onDataChange(DataSnapshot dataSnapshot) {

                if (!isNotificationSent) {

```

```

        String message = dataAppointment.getRequesterName()
            + " has requested an appointment request to " +
dataAppointment.getResponderName();

        new SendNotification(Constant.PROFESSOR,
            dataProfessor.getProfessorDeviceTkn(), message);
        // saveNotificationToFirebase(dataAppointment, message);
    }

    Utils.displayToast(context.getString(R.string.appointment_requested),
context);

    ((BookAppointment) context).finish();
}

@Override
public void onCancelled(DatabaseError databaseError) {
    Utils.displayToast(context.getString(R.string.something_went_wrong),
context);
}

});
} else {
    Utils.displayToast("There is no time slot available on this day", context);
}
} catch (Exception ex) {
    Utils.displayToast(context.getString(R.string.something_went_wrong),
context);
    ex.printStackTrace();
}

} catch (Exception ex) {
    ex.printStackTrace();
}
}

public String getKey(int date) {

    String key = "";
    switch (date) {
        case 2:
            key = "MON";
            break;
        case 3:
            key = "TUE";
            break;
        case 4:

```

```

        key = "WED";
        break;
    case 5:
        key = "THU";
        break;
    case 6:
        key = "FRI";
        break;
    case 7:
        key = "SAT";
        break;
    case 1:
        key = "SUN";
        break;
    }
    return key;
}

private ArrayList<String> getBusyHours(String key) {

    ArrayList<String> busyHours = new ArrayList<>();

    if (dataProfessor.getBusyHours() != null)
        if (dataProfessor.getBusyHours().get(key) != null)
            return dataProfessor.getBusyHours().get(key);
    return busyHours;
}

private ArrayList<String> getFreeHours(String key) {

    ArrayList<String> freeHours = new ArrayList<>();

    if (dataProfessor.getFreeHours() != null)
        if (dataProfessor.getFreeHours().get(key) != null)
            return dataProfessor.getFreeHours().get(key);
    return freeHours;
}

```