MINISTRY OF EDUCATION
IMAM ABDULRAHMAN BIN
FAISAL UNIVERSITY
COLLEGE OF COMPUTER SCIENCE
& INFORMATION TECHNOLOGY

وزارة التعليم
جامعة الإمام
عبدالرحمن بن فيصل
كلية علوم الحاسب
وتقنية المعلومات

جامعة الإمام عبد الرحمن بن فيصل
IMAM ABDULRAHMAN BIN FAISAL UNIVERSITY

CS 516 Advanced Programming Language

(Term 2, 2019-2020)

APL Report

The Bump Project

Report Submitted to:

Mrs. Hanoof Algofari, CCSIT-IAU

Submitted by:

Name	ID
Fay Alabdulhadi	2160004777
Rahaf Saleh Alzahrani	2160006662
Tasneem Hamdy Dosoqi	2160007430
Muneera Abdulrahman Alshaalan	2160005828
Khawlah Fahad Bajbaa	2160003191

Date: Sunday April 5, 2020

Declaration Statement:

We herby declare that this project “The Bump” is based entirely on our hard work as a team. No aspect of this project was made or created by anyone else. All codes that are included are written by us, inspired from the curriculum’s slides and book, and with the aid of online sources.

Introduction / Problem Statement:

Our application is a pregnancy and baby tracking application that helps soon to be mothers in their pregnancy journey. It allows them to follow their child’s development and growth inside the womb and get weekly updates about their pregnancy such as their baby size, upcoming appointments and milestones. It also allows the soon to be mothers to understand how their body changes as the pregnancy progresses, connect with a community of soon to be moms and book important appointments and events.

Project Goals and Objectives/Deliverables:

User (Mom) Tasks	*Admin (Expert) Tasks*
• Create a profile for the mom • Viewing and editing the profile • View the baby’s current size and get information about baby’s growth • View the Frequently Asked Questions (FAQ) • Post questions related to pregnancy • Add/Delete/Modify appointments and important milestones	• Answer questions that moms post • Post material about mom and baby growth (weekly updates) • Delete / Modify material about mom and baby growth (weekly updates) • Viewing and editing the profile

Project Scope:

Creating 2 applications, one written in swift programming language for iOS users, and another one written in java programming language for Android users. There are 2 types of main users involved which are:

- Regular user (Mom)
- Admin (Expert)

Success Factors and Benefits:

Success Factors:

- Achieve project's objectives.
- Produce Error-free systems.
- Submit the milestones on time.
- Using the cloud services to manage the teamwork.
- The systems should have flexibility and adapt to changes.
- Each team member contributes their fair share of the workload and fully understands what their responsibilities are and where they fit in.

Benefits:

- Different Android App components are applied on The Bump App, such as activities, fragments, content provider, and intents.
- Learned how to make the same activity work for both admin and the normal user
- Learned how to make smart code.
- Retrieved all the knowledge that is related to Mobile App System course when we implemented the java language.

Limitations/Restrictions:

Considering the time limit of the project, there had to be a limitation which is having updates of the mother and the baby that based on a monthly basis instead of the normal weekly basis.

Selected programming language 1:

Language: Swift

Programming paradigm: multi-paradigms.

It allows imperative programming between Object Oriented or procedural programming or declarative paradigm or functional programming. You can refer to the source code in the last page of this report.

Starting Page:

The first page the user will see, is the splash page. The user can log in by clicking the sign in button if she already has an account or can click the sign-up button if she doesn't have an account yet.

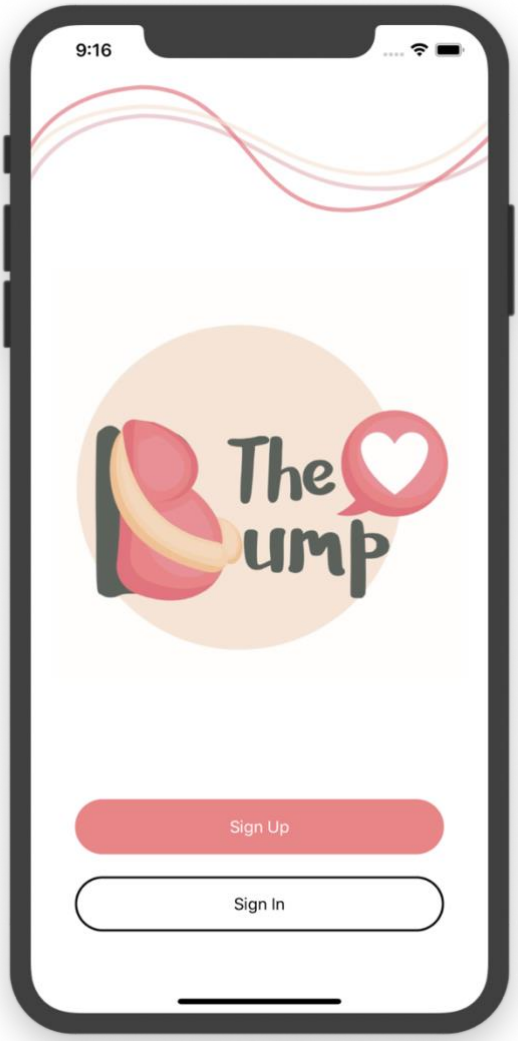


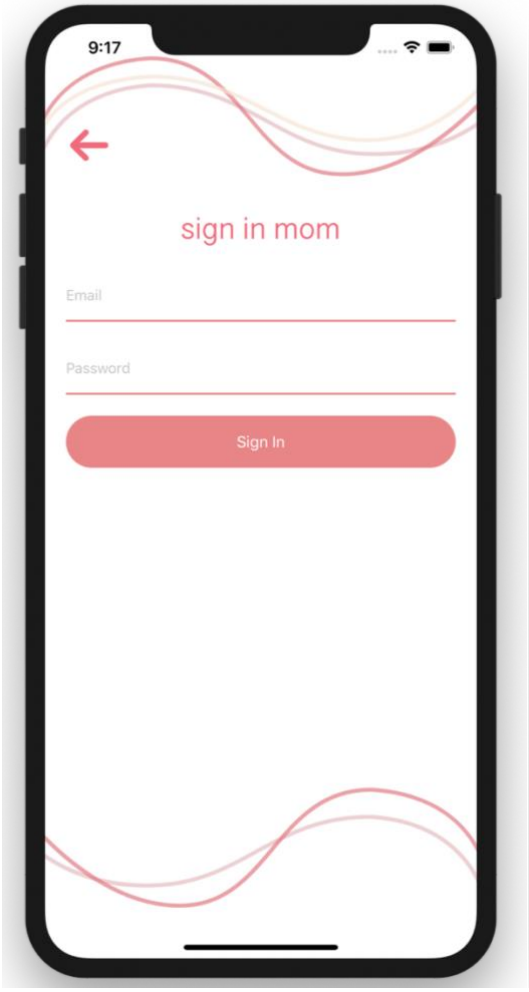
Figure 1 Starting Page

Login Page:

The user can login to the application by entering a valid username and password for an account that already exists. This interface contains 2 fields: these fields enable the user to enter her username and password.

When the user clicks on (Sign In) button after filling all fields with the correct username and password, the user will be able to login to the application. On the other hand, when the user clicks on (Sign In) button without filling the fields or when the user enters an

incorrect username and password, the red error message will directly appear under the (Sign In) button.



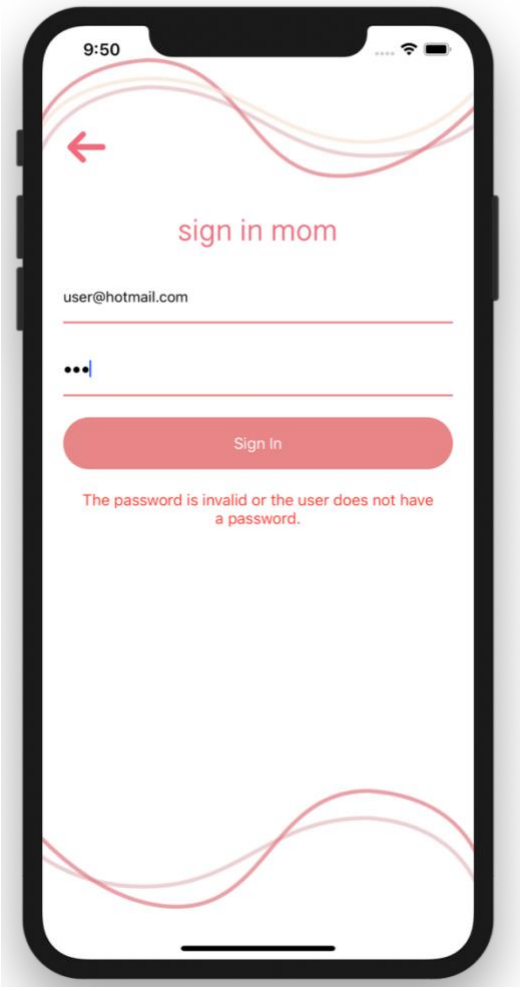


Figure 2 Sign In Page

Figure 3 Invalid User Or Password

In addition, if the user does not have an account in this app, then she can create a new account by clicking on (Back Arrow) button that will take them back to the starting screen, then she can click on (Sign Up) button.

Sign Up Page:

In this page, the user can create an account in this app by filling all fields that appear in Figure 4 below. When the user clicks on (Sign up) button with filling all fields then a new user account will be created, and all this information will be added in the database.

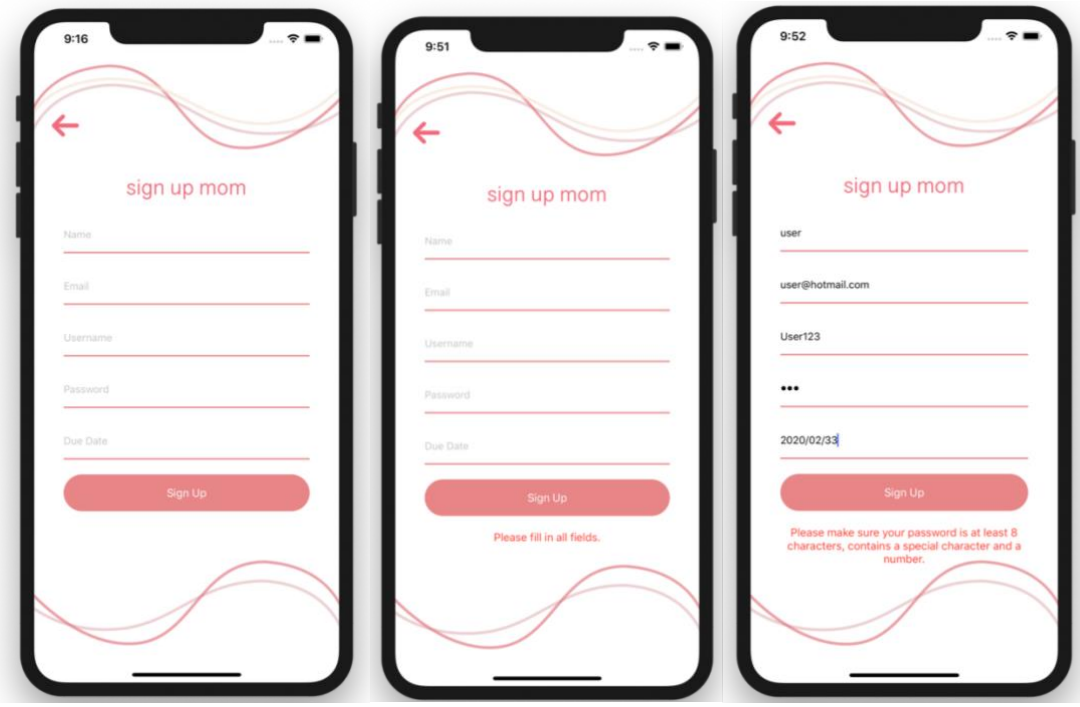


Figure 4 Sign Up Page *Figure 5 All field checking.* *Figure 6 Weak Password Checking*

When the user clicks the (sign up) button, the system will check if the user writes all the required fields. If not, error message will appear under the (sign up) button.

In addition, the system will not accept any weak passwords, the user must write a strong password including capital and small letters, numbers and special character too.

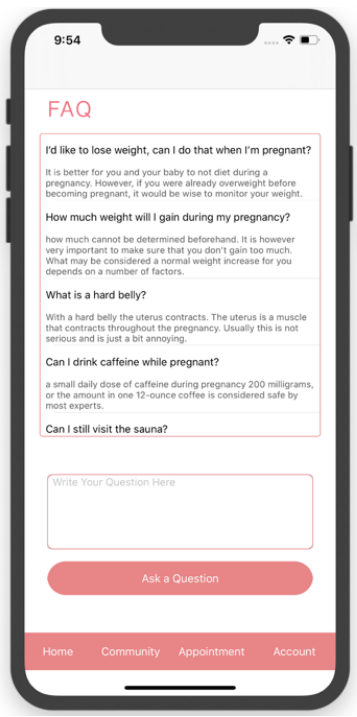
Also, if the user already has an account in this app, then she can log in to her account by clicking on (Back Arrow) button that well back to the starting screen, then she can click on (Sign In) button.

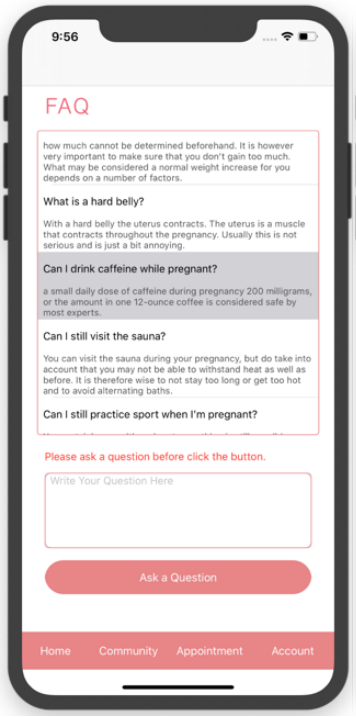
Frequently Asked Questions (FAQ) Page (User):

In this page, the user can review the FAQ, which is a list of common questions people have asked about pregnancy. In addition, if the user has a question that has never been asked before and the user did not find it in the FAQ list, then the user has the ability to ask a question by filling the new question field that appears in figure 7.

When the user clicks on (Ask a Question) button without filling the question field first, then the system will check if the question filed is empty, if it is, an error message will appear above the question field as shown in figure 8.

The other scenario is when the user clicks on (Ask a Question) button after filling the question field, then the system will retrieve the question being asked by the user from the question field and it will add it to the Firebase Cloud Firestore database in the collection (AskedQuestions), then an alert message dialog will be shown as in figure 9, with a message to inform the user that the question will be answered soon.





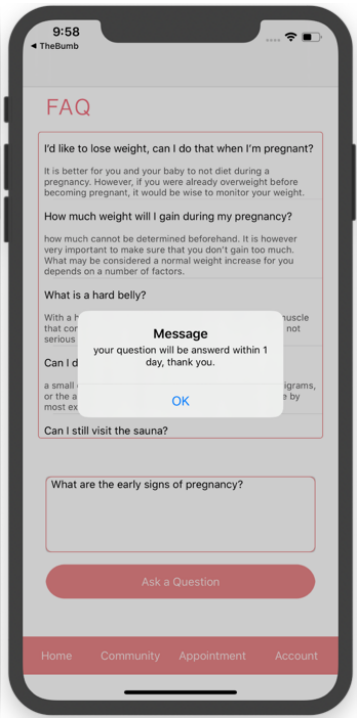


Figure 7: FAQ Page (User)

Figure 8: Question Field Checking.

Figure 9: Alert Message Dialog.

Frequently Asked Questions (FAQ) Page (Admin):

In this page, the admin can review all the asked questions about pregnancy that need an answer as shown in figure 10. The admin has the ability to navigate through all questions in the list. In order to answer a question, the admin first needs to choose any question from the list, the system will fill the question label under the list with the chosen question that the admin chose from the list. In this page there are multiple scenarios:

- When the admin clicks on (Answer Question) button without choosing a question first, an alert message dialog will be shown as in figure 11 with a message to inform the admin that he/she needs to choose a question first.
- When the admin clicks on (Answer Question) button after choosing a question and without filling the answer field first, then an alert message dialog will be shown as in figure 12 with a message to inform the admin that he/she needs to answer the chosen question first.
- When the admin clicks on (Answer Question) button after choosing a question and after filling the answer field, the system will retrieve the question being asked from the question label and the answer that was written by the admin from the answer filed and it will add it to the Firebase Cloud Firestore database in the collection (Community), finally an alert message dialog will be shown as in figure 13 with a message to inform the user that the question and its answer will be added to FAQ user page soon.
- After the admin answers a particular question, that question will be removed from the question list and the list will be updated as shown in figure 14 below to avoid answering the same question many times.
- When the admin clicks (back) button, the system goes back to the admin account page.

Figure 10: FAQ Page (Admin)

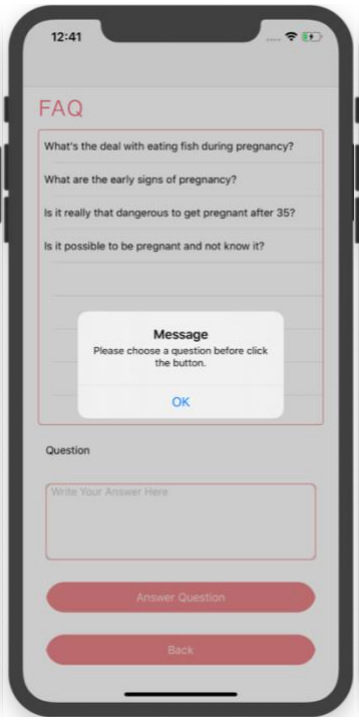


Figure 11: Question Label Checking.

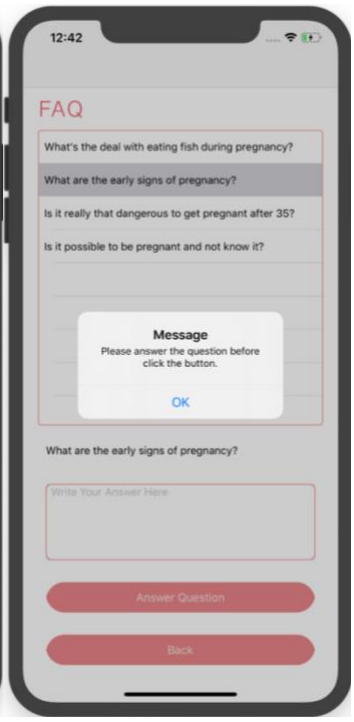


Figure 12: Answer Field Checking.

Figure 13: Alert Message Dialog.

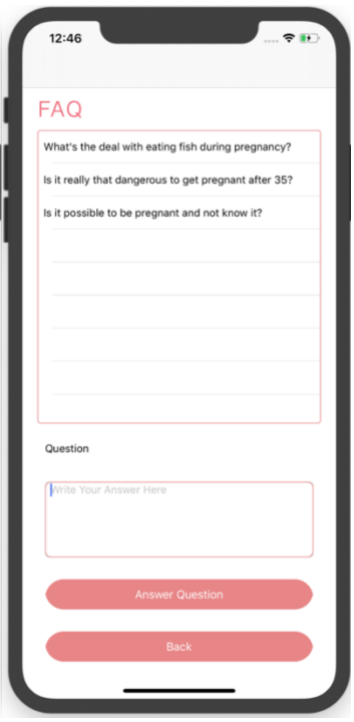


Figure 14: List After Updating.

Home Page:

This page is the main page of the application. It consists of a summary of the pregnancy developments during the current month the expecting mother is experiencing. It shows an image of what the mother's body looks like with the fetus, a description and image of the baby's current size, and finally information about the baby's development inside the womb. The page changes according to the current month. By using the navigation bar below, the user can move around the rest of the pages of the application.

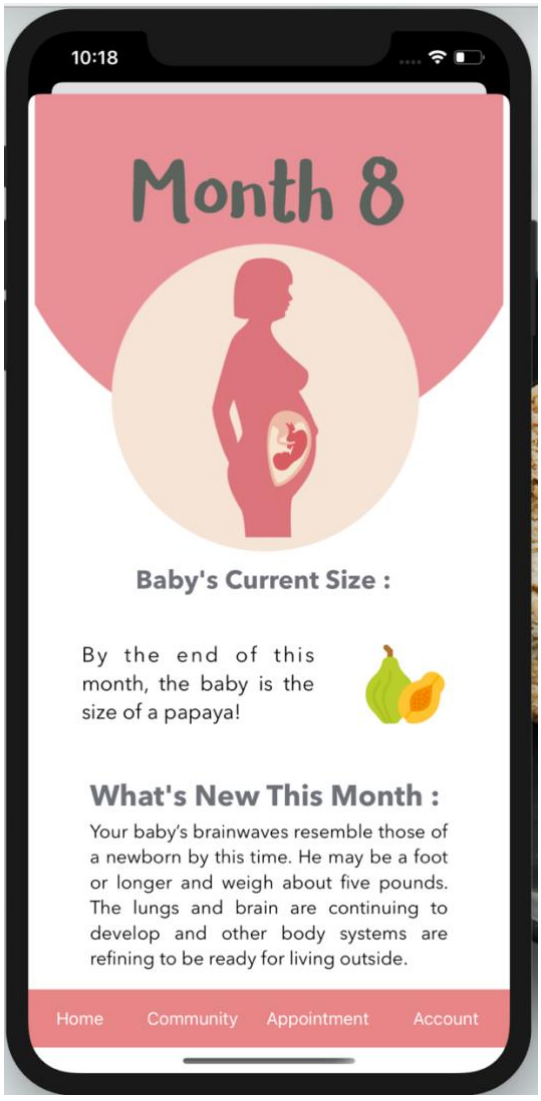


Figure 15: Home page 1

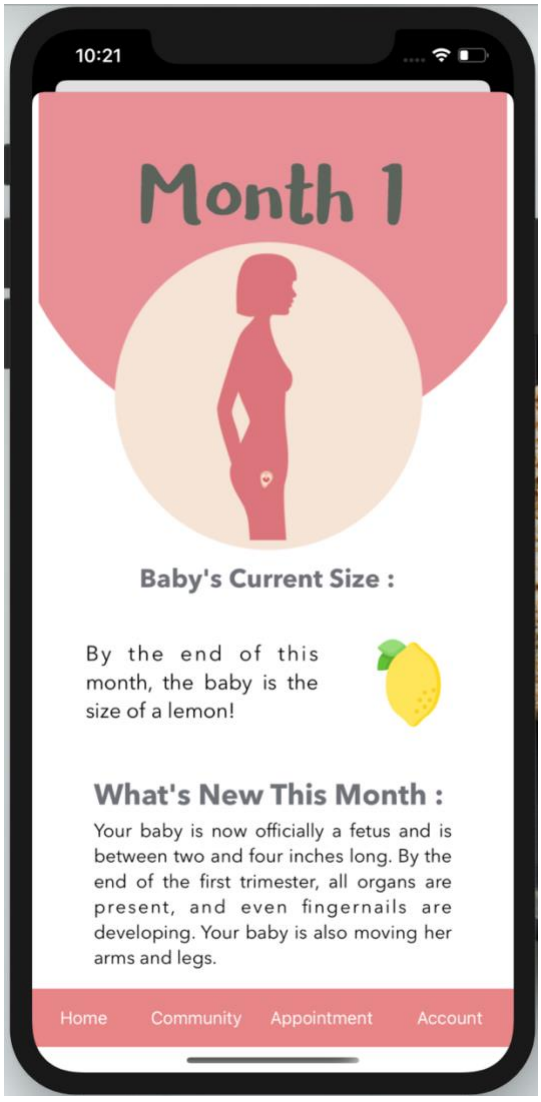


Figure 16: Home page 2

Admin and User Account Page:

This page displays a summary of the current user's information such as name, user type, password, date (for regular users). In addition, this page enables the user and admin to navigate to the editing page by clicking on the button "Modify". The signout button enables the user to log out of their account.

For the Admin, this page is accessed through the home and login page, if it is verified that the entered email is an official email, the admin account page will appear. For User, this page is accessed through the navigation bar, by clicking on "Account".

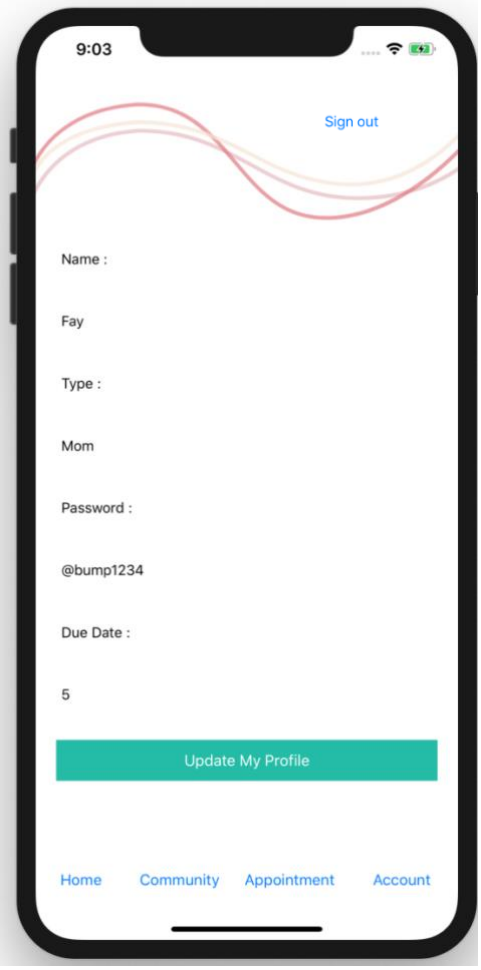


Figure 15: User Account page.

Figure 16: Admin Account page.

Admin and User Edit Account Page:

This page enables the user to easily modify their data by filling in the fields to be modified. The program checks the user's entries and displays error messages in case the entries are not approved, then modifies them.

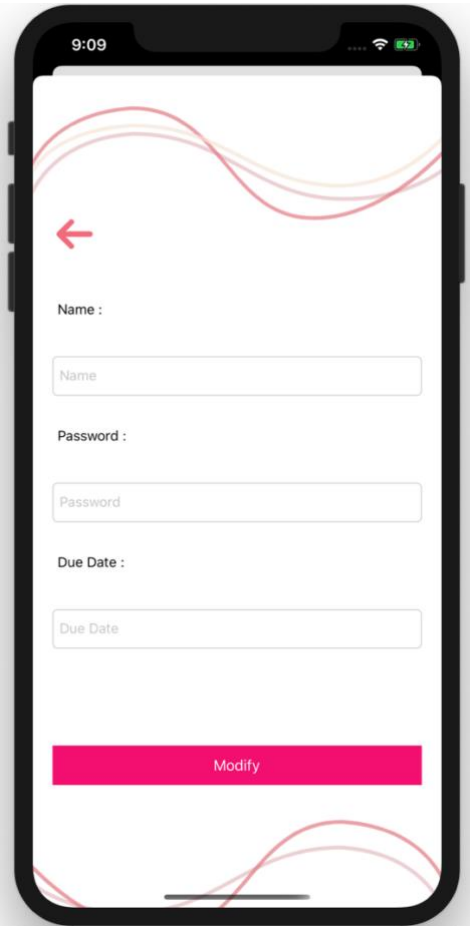


Figure 17: User Edit page.

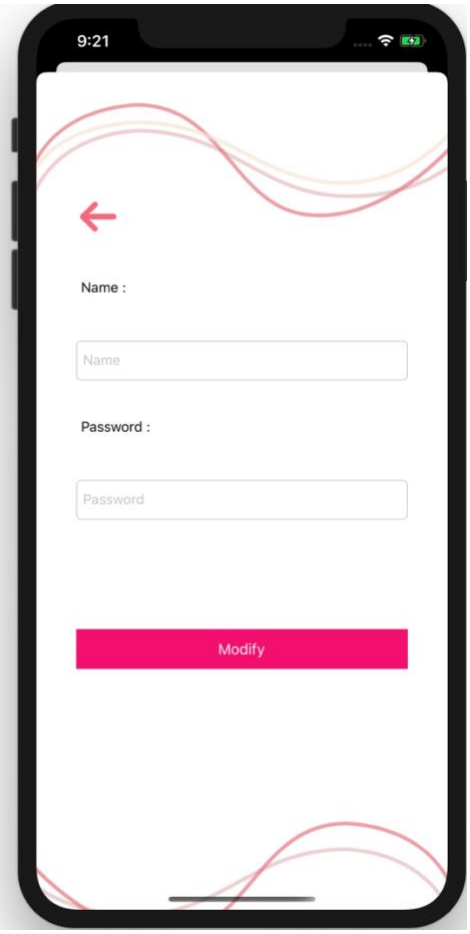


Figure 18: Admin Edit page.

The images below represent the account data modification process for the user (same case with admin) by changing the name to "123", the password to "89h" and the date to "hgk" and shows error massages according to the situations. After entering valid input like "MuneeraShaalan", the update will happen and the program navigates back to the account page where the data has been modified.

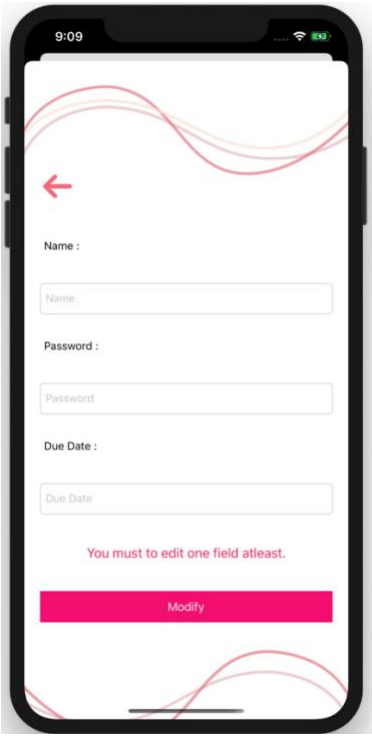


Figure 19: Error 1 .

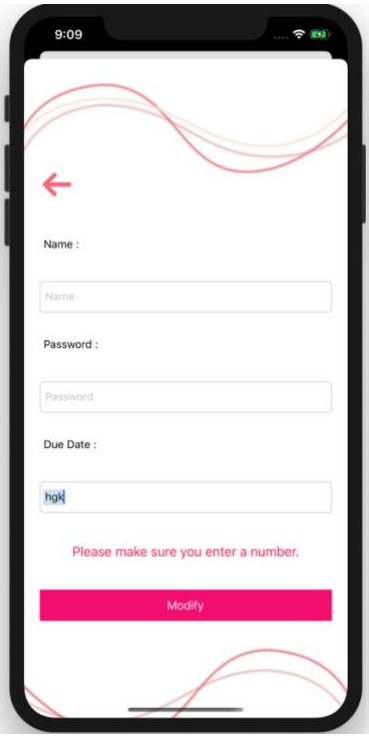


Figure 20: Error 2.

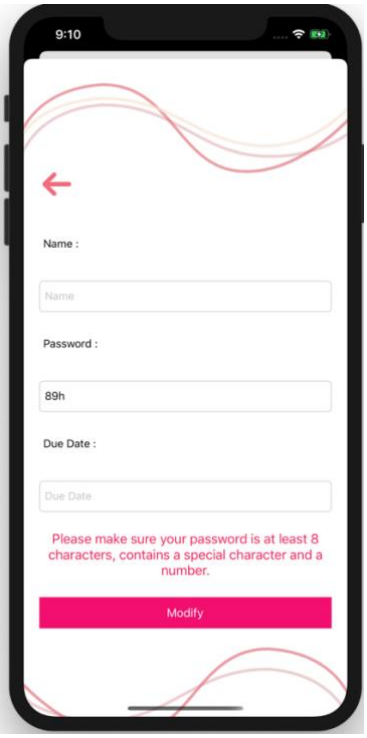


Figure 21: Error 3.

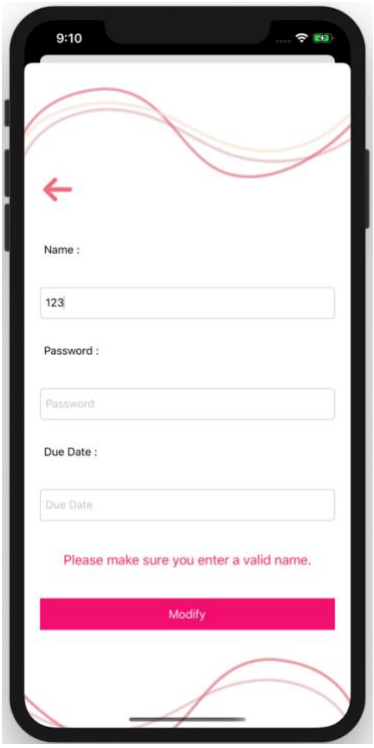


Figure 22: Error 4.

Figure 23: Updated.

Appointments Page:

In this page, the user can view the upcoming appointments for him/her as shown in figure 26. Also, the user can add pregnancy appointments by clicking on the (+) button and the dialog will appear as shown in figure 27. When the user clicks on add button, the added appointment will appear in the checklist view. They can also click on cancel then the dialog will disappear, and nothing will happen. If the appointment was added, the user can mark it as done when the appointment is done as show in figure 28, or she/he can delete it by swiping the appointment name to the left as shown in figure 29.

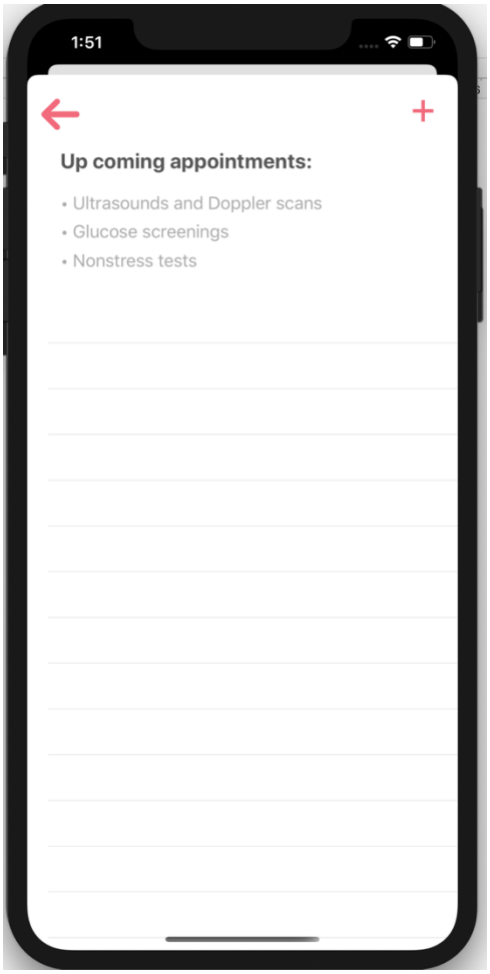


Figure 24 Appointments page

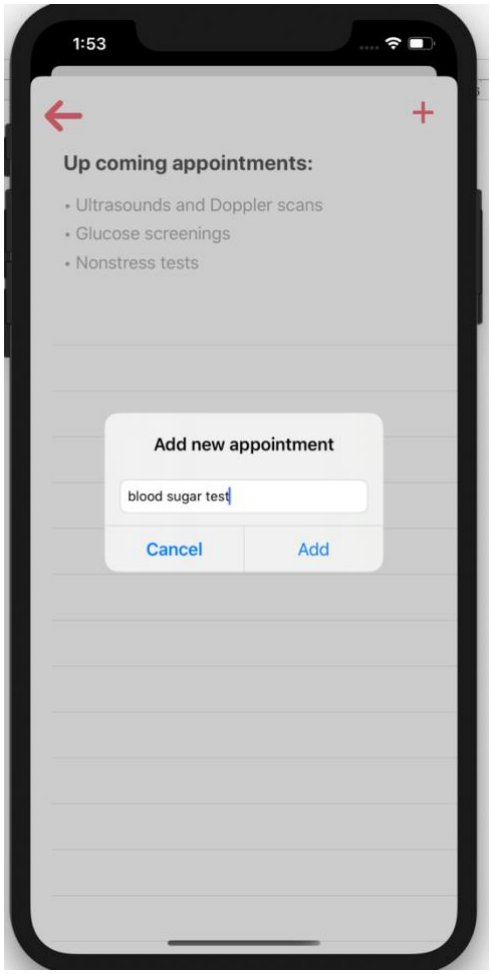


Figure 25 Add appointment dialog

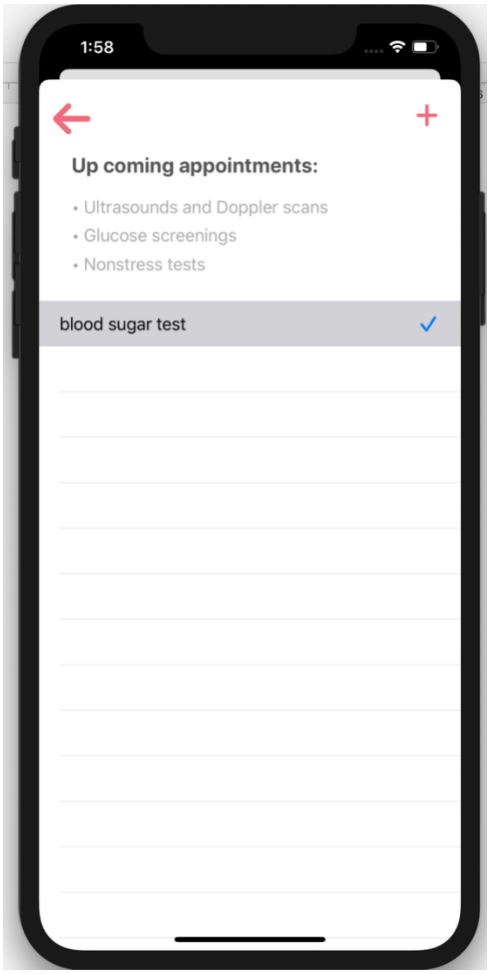


Figure 28 checked appointment

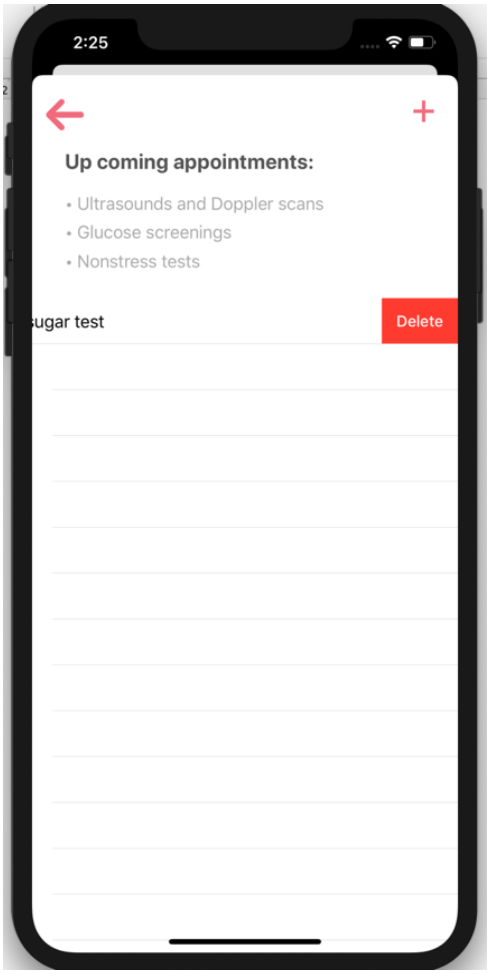


Figure 29 delete appointment

Selected programming language 2:

Language: Java

Programming paradigm: Object oriented

Here we use the same functionality that we developed by XCode (Swift) but by using anther programing language which is android (Java). You can refer to the source code in the last page of this report.

Splash Screen:

The first page the user will see, is the splash page. it will be automatically disappeared after 3 seconds and the sign in page will be started.

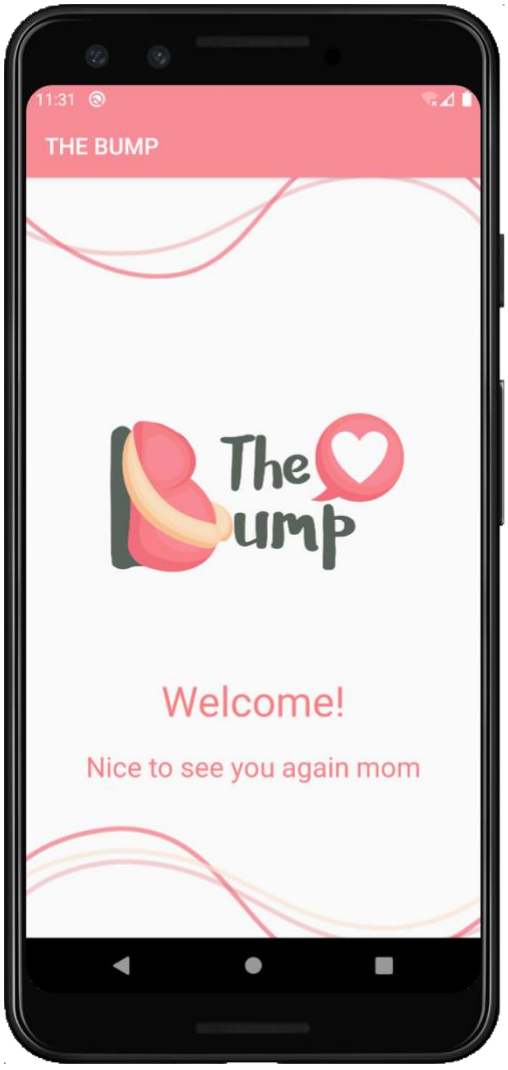


Figure 30 Splash Screen

Sign in Page:

The user can login to the application by entering a valid username and password for an account that already exists. This interface contains 2 fields as previously: username and password.

When the user clicks on (Sign In) button after filling all fields with the correct username and password, the user will be able to login to the application and go to her Account page. On the other hand, when the user clicks on (Sign In) button without filling the fields or when the user enters an incorrect username and password, the error message will directly appear.

When the user clicks the (sign up) button the system will check if the user writes all the required fields. If not, error messages will appear under the (sign up) button.

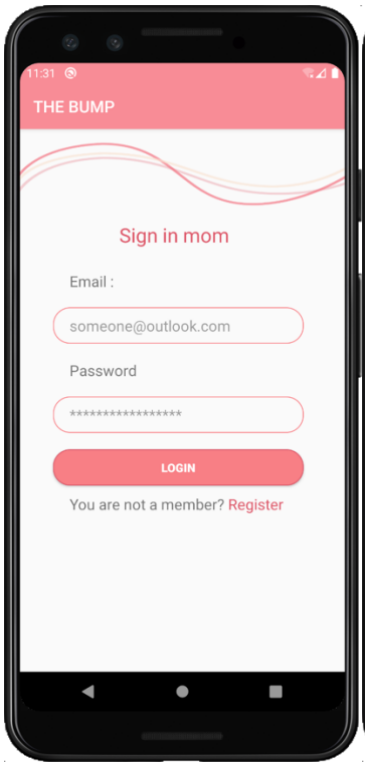


Figure 31 Sign in Page

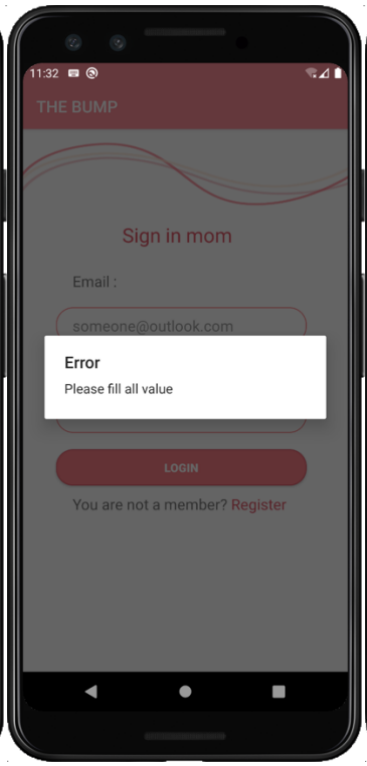


Figure 32 All field checking

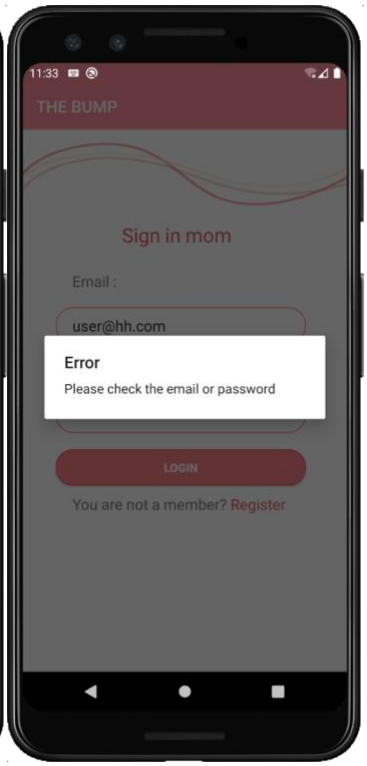


Figure 33 Invalid User Or Password

In addition, if the user does not have an account in this app, then she can create a new account by clicking on (Register) link that appears in the bottom.

Sign Up Page:

In this page, the user can create an account in this app by filling all fields that appear in figures below. When the user clicks on (Sign up) button after filling all fields, then a new user account will be created, and all this information will be added in the database.

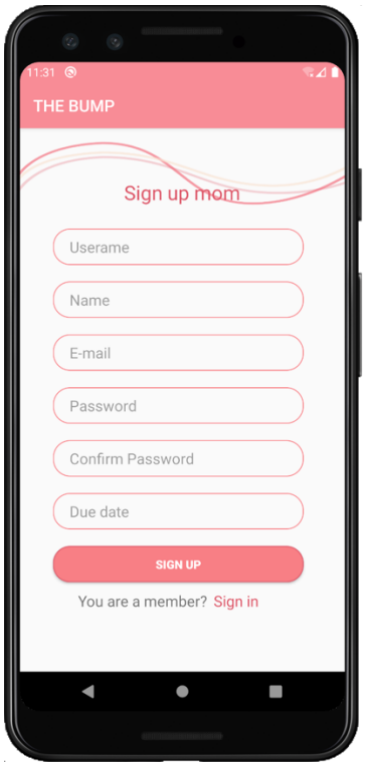


Figure 34 Sign Up Page.

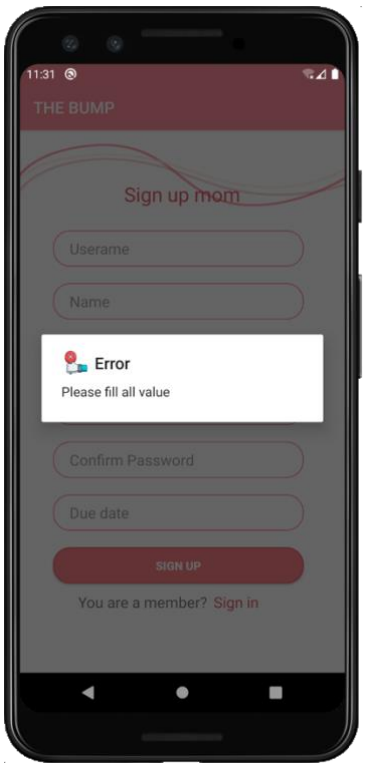


Figure 35 All field checking

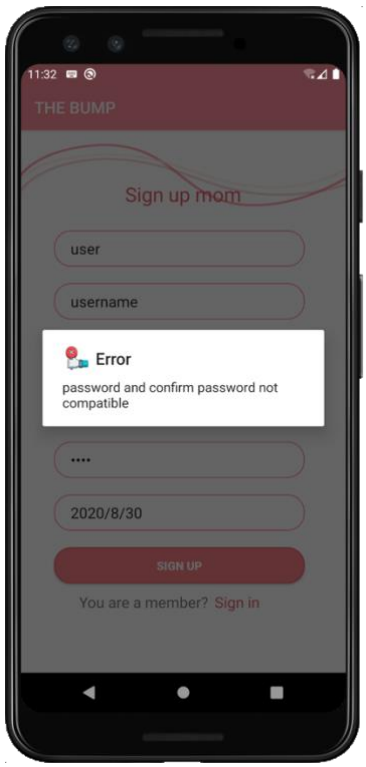


Figure 36 Confirmation Password Error

When the user clicks the (sign up) button, the system will check if the user writes all the required fields. If not, error message will appear clarifying the error.

In addition, the system will check if the password and the confirm password field are similar. If not, error messages will appear to clarify the error to the user.

Also, if the user already has an account in this app, then she can log in to her account by clicking on (Sign in) link that appears in the bottom.

Frequently Asked Questions (FAQ) Page (User):

In this page, the user can review the FAQ, which is a list of common questions people have asked about pregnancy. In addition, if the user has a question that has never been asked before and the user did not find it in the FAQ list, then the user has the ability to ask a question by filling the new question field that appears in figure 37.

When the user clicks on (Ask a Question) button without filling the question field first, then the system will check if the question field is empty, if it is, an error message will appear as shown in figure 38.

The other scenario is when the user clicks on (Ask a Question) button after filling the question field, then the system will retrieve the question being asked by the user from the question field and it will add it to the SQLite database in the table (AskedQuestions), then an alert message dialog will be shown as in figure 39 with a message to inform the user that the question will be answered soon.

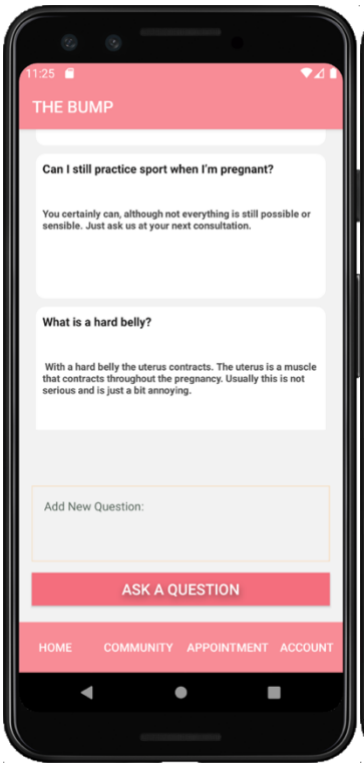


Figure 37:FAQ Page (User)

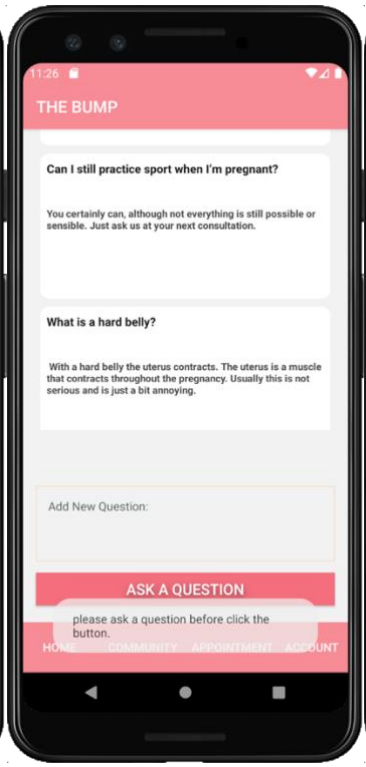


Figure 38: Question Field Checking.

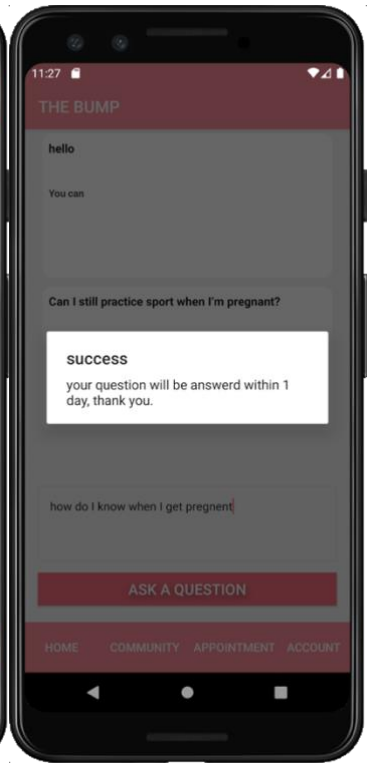


Figure 39: Alert Message Dialog.

Frequently Asked Questions (FAQ) Page (Admin):

In this page, the admin can review all the asked questions about pregnancy that need an answer as shown in figure 40. The admin has the ability to navigate through all questions in the list. In order to answer a question, the admin first needs to choose any question from the list, the system will fill the question label under the list with the chosen question that the admin choose from the list. In this page there are multiple scenarios:

- When the admin clicks on (Answer Question) button without choosing a question first, then an error message will be shown as in Figure 41 below with a message to inform the admin that he/she need to choose a question first.
- When the admin clicks on (Answer Question) button after choosing a question and without filling the answer field first, an error message will be shown as in figure 42 below with a message to inform the admin that he/she needs to answer the chosen question first.
- When the admin clicks on (Answer Question) button after choosing a question and after filling the answer field, the system will retrieve the question being asked from the question label and the answer that written by the admin from the answer filed and it will add it to the SQLite database in the table (Community), then an alert message dialog will be shown as in figure 43 with a message to inform the user that the question and its answer will be added to FAQ user page soon.
- After the admin answers a particular question, that question will be removed from the question list and the list will be updated as shown in figure 44 to avoid answering the same question many times.
- When the admin clicks the (back) button, the system goes back to the admin account page.

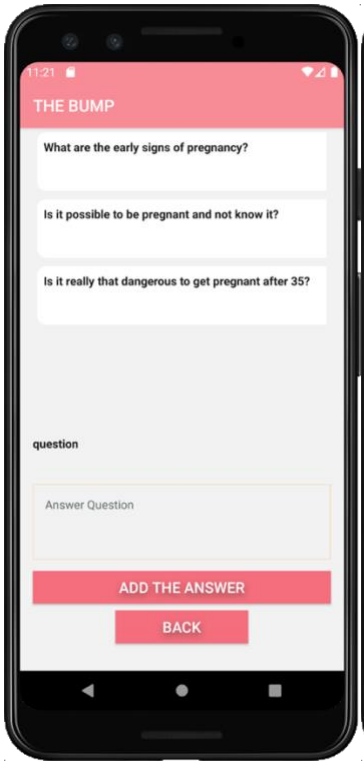


Figure 40:FAQ Page (Admin)

Figure 41:Question Label Checking.

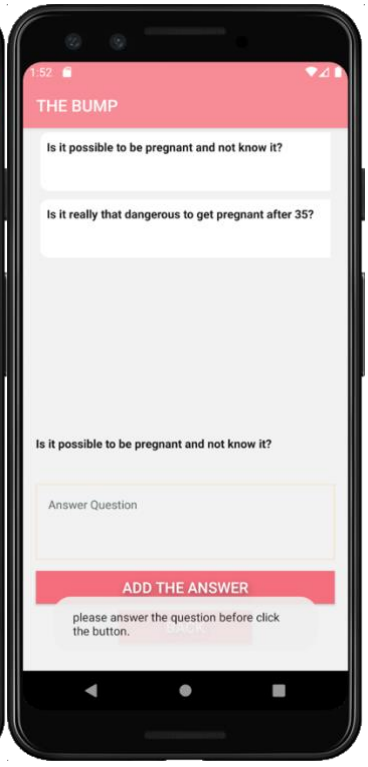


Figure 42: Answer Field Checking.

Figure 43:Alert Message Dialog.

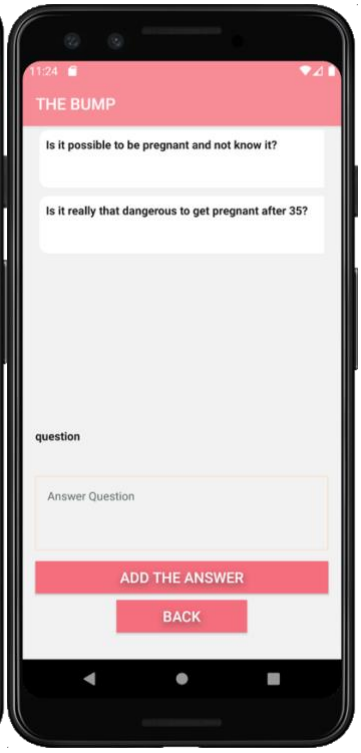


Figure 44: List After Updating.

Home Page:

This page is the main page of the application. It consists of a summary of the pregnancy developments during the current month the expecting mother is experiencing. It shows an image of what the mother's body looks like with the fetus, a description and image of the baby's current size, and finally information about the baby's development inside the womb. The page changes according to the current month. By using the navigation bar below, the user can move around the rest of the pages of the application.

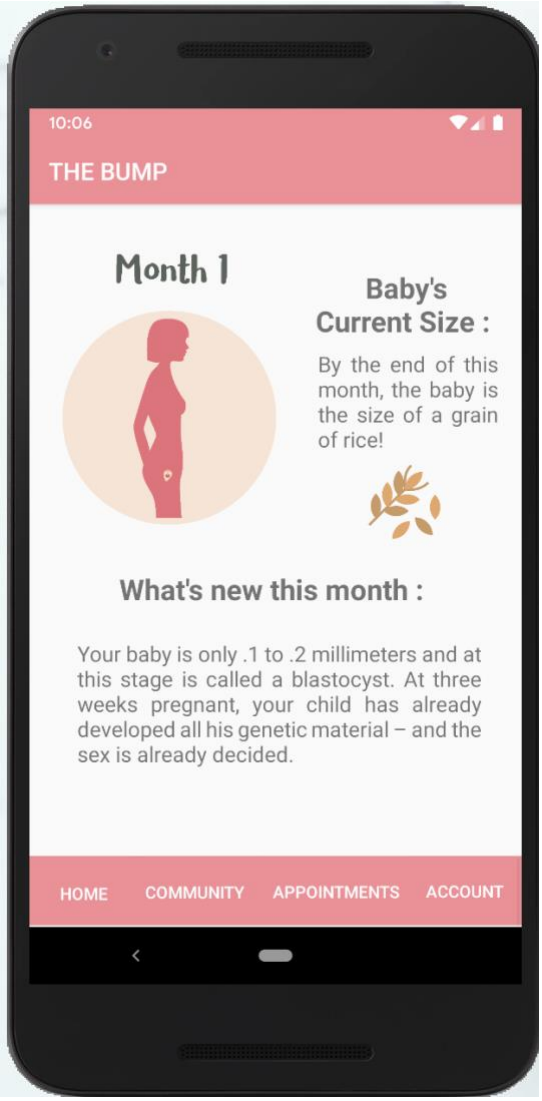


Figure 45 : Home page 1

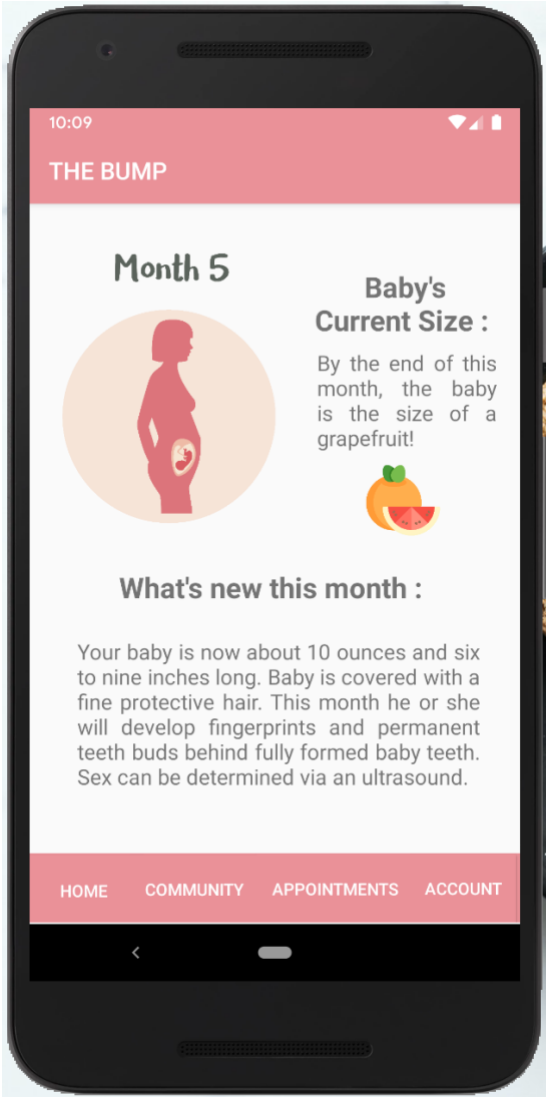


Figure 46 : Home page 2

Admin and User Account Page:

This page displays a summary of the current user's information such as name, user type, password, date (for regular users). In addition, this page enables the user and admin to navigate to the editing page by clicking on the word "Modify". By clicking on the signout button, the user can log out.

For the Admin, this page is accessed through the login page, if it is verified that the entered email is an official email, the admin account page will appear. For the User, this page is accessed through the navigation bar, click on "Account", then the user account page will appear.

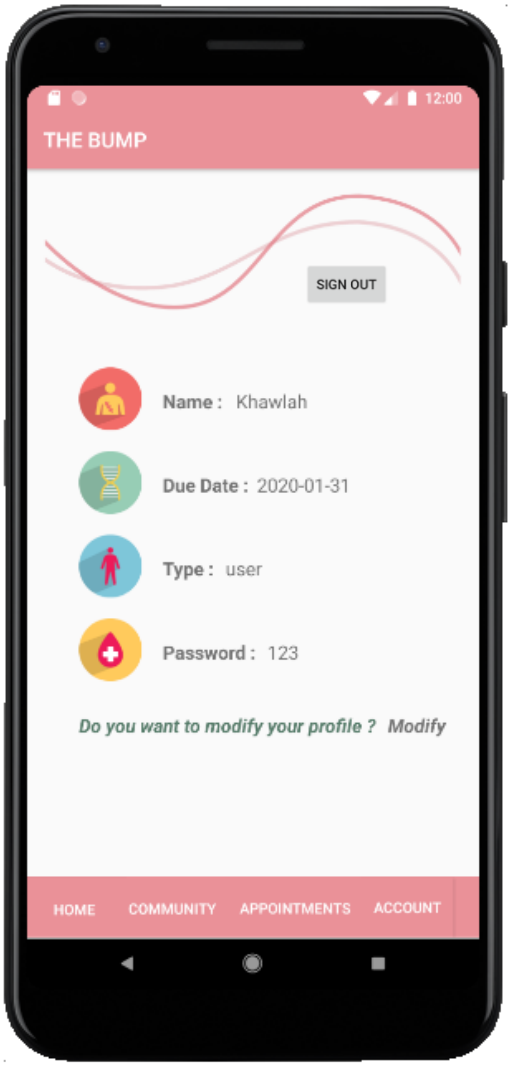


Figure 47: User Account page.

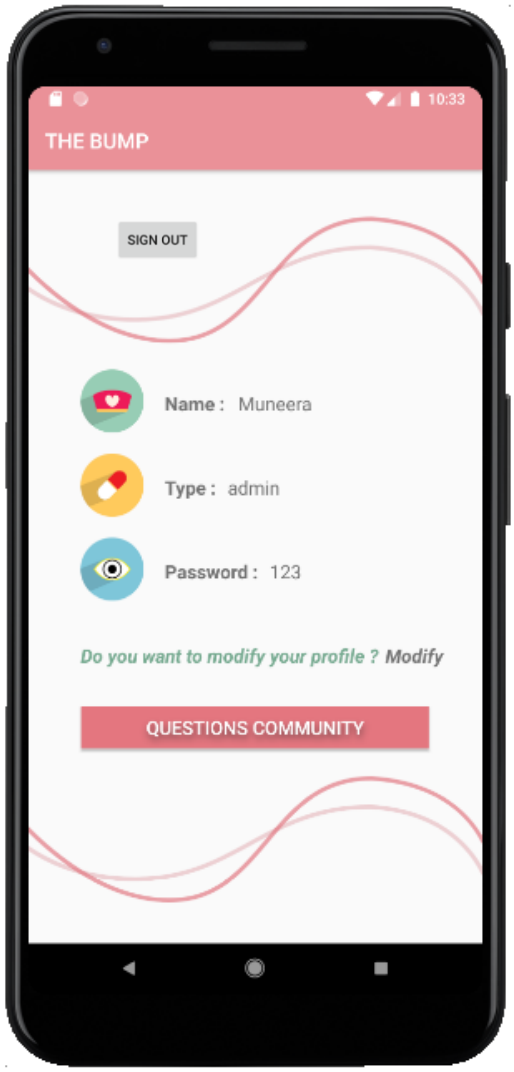


Figure 48: Admin Account page.

Admin and User Edit Account Page:

This page enables the user to easily modify their data by filling in the fields to be modified. The program checks the user's entries and displays messages in case the entries are not approved, then modifies them.

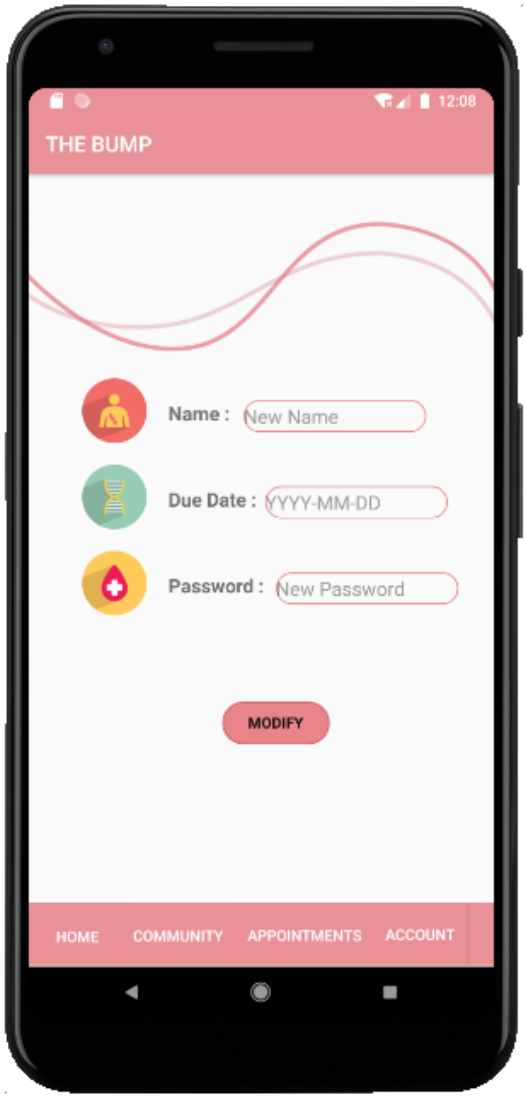


Figure 49: Edit User page.

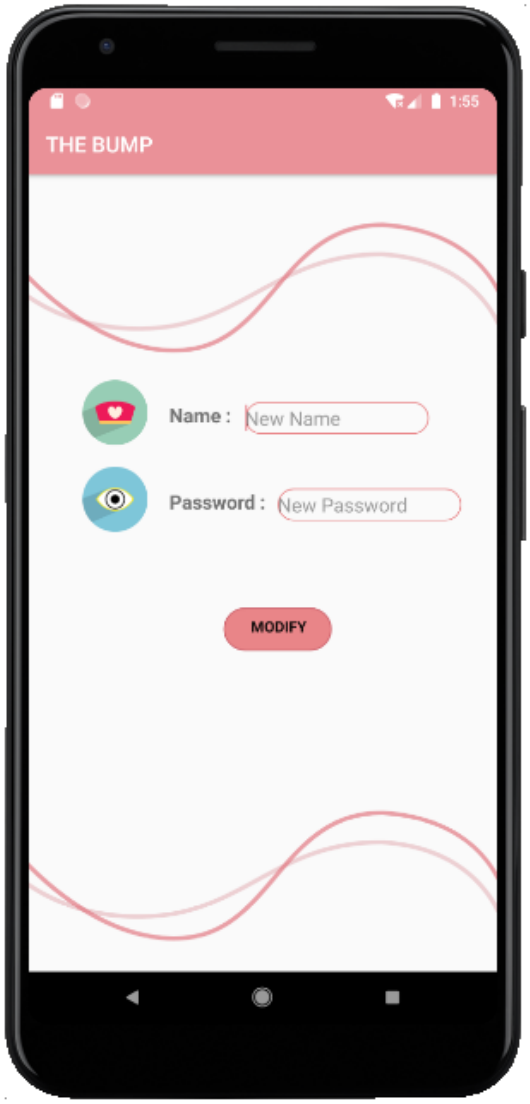


Figure 50: Edit Admin page.

The images below represent the account data modification process for the admin by changing the name from “Muneera” to “Muneera Shaalan” and the password from “123” to “12345”, then navigating to the account page where the data has been modified.

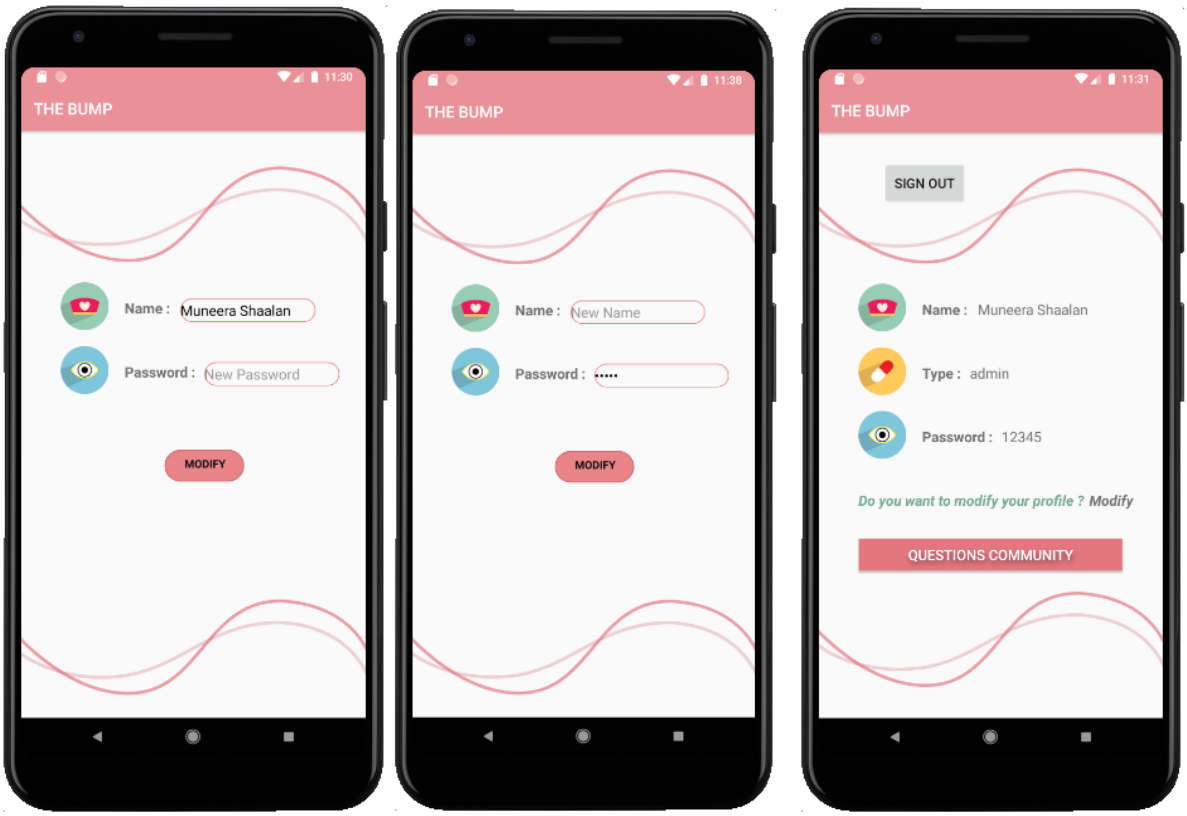


Figure 51: Edit Admin name. *Figure 52: Edit Admin password.* *Figure 53: After Editing.*

The figures below represent the account data modification process for the user by changing the name from “Khawlah” to “Khawlah Fahad” and the date from “2020-01-31” to “2020/1/30” which shows an error message that says “The date must be in YYYY-MM-DD format ”, then changes it to “2020-01-30” . After that, navigating to the account page and the data has been modified.

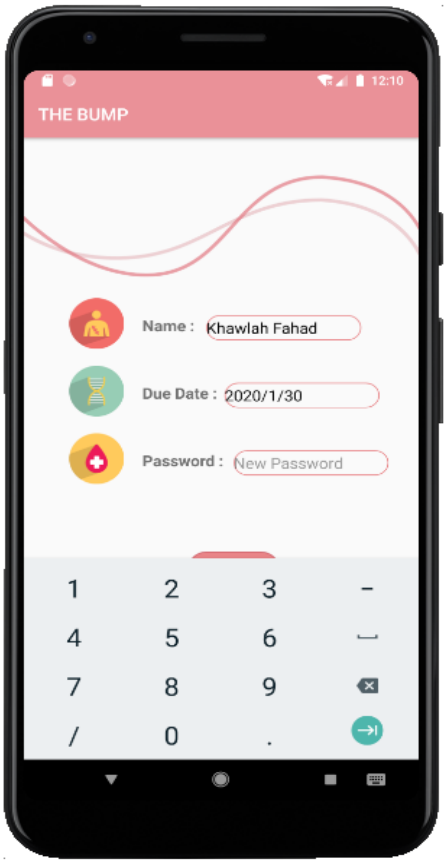


Figure 54: Edited with wrong date.

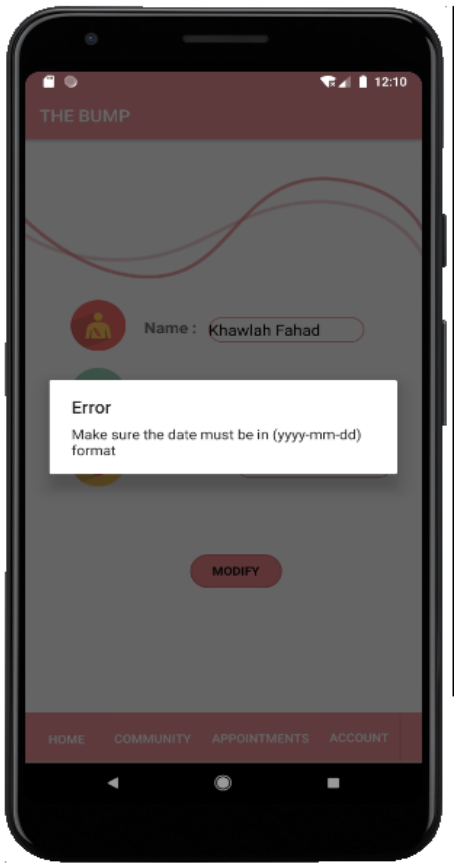


Figure 55: Error massage.

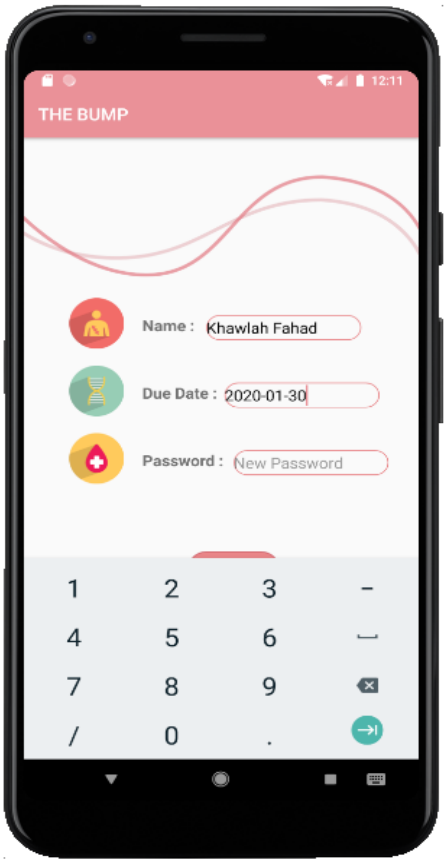


Figure 56: Valid date.

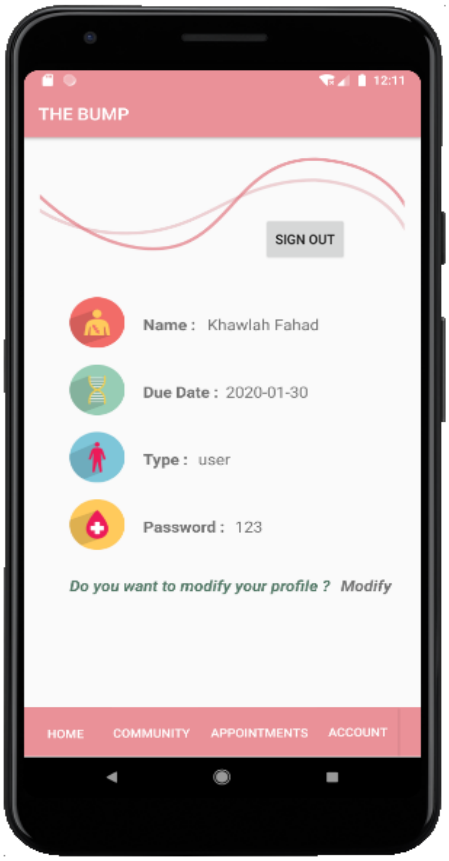


Figure 57: After Editing.

Appointments Page:

In this page, user can view the upcoming appointments for him/her as shown in figure 58. Also, the user can add pregnancy appointments by clicking on the (+) button and the dialog will appear as shown in figure 59. When the user clicks on add button, the added appointment will appear in the checklist view or if they click on cancel then the dialog will disappear, and nothing will happen. If the appointment was added, the user can mark it as done when the appointment is done as show in figure 60, or she/he can delete it by clicking on the (X) button.

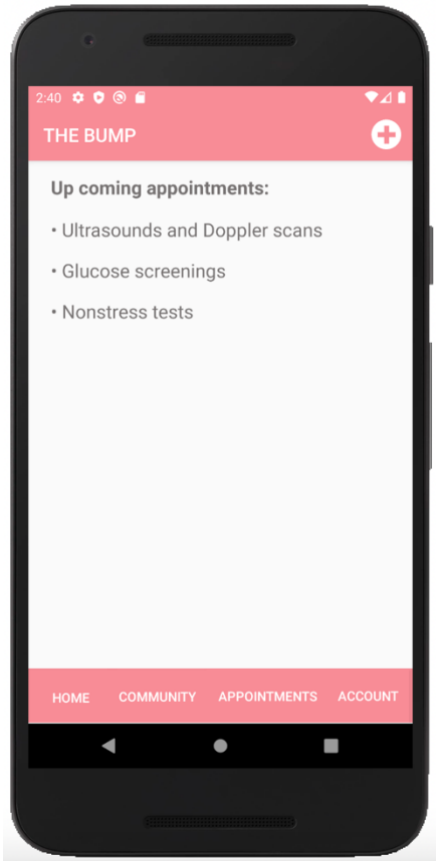


Figure 58 Appointments page

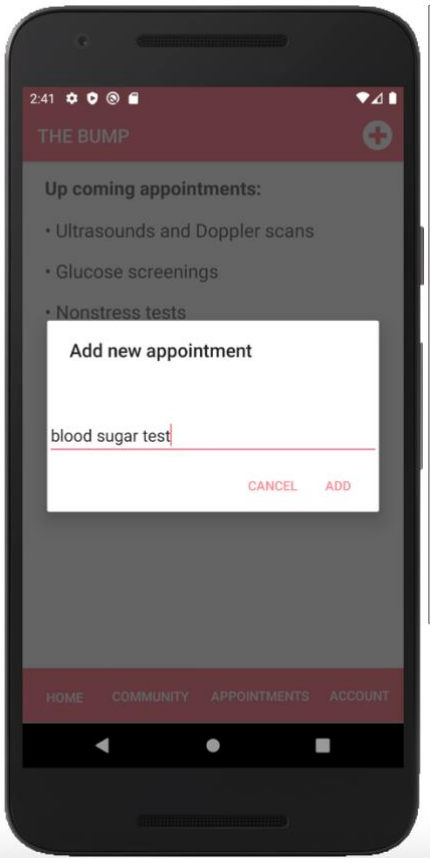


Figure 59 Add appointment dialog

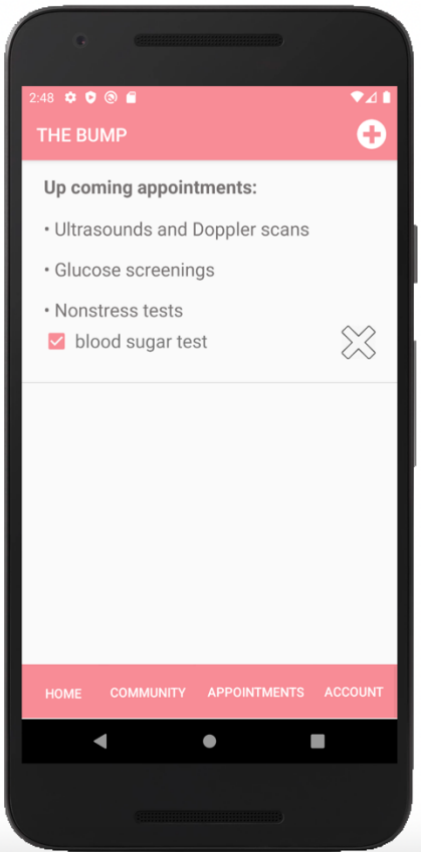


Figure 60 checked appointment

Comparison between two programming languages:

SYNTAX:

The following is a simple syntax comparison between Swift and Java:

Hello, world!

Swift:

```swift
print("Hello, world!")
```

Java:

```java
public class HelloWorld {

    public static void main(String[] args) {
        System.out.println("Hello, world!");
    }

}
```

As you can obviously see, it's a lot more code in Java and Objective-C than in Swift. This single line in Swift tells us a lot: Swift doesn't need a main function and semicolons at the end of expressions and works also like a script language. If we only want to do something simple like this print statement, languages like Java produce an overhead, because we are forced to create a class.

Type inference

Swift

```swift
let e: Int = 3
var a = 5.678; // implicit Double

// a = a + e
a = a + Double(e)

print("\(e) \(a)")
```

Java

```java

    public static void main(String[] args) {
        final int e = 3;
        double a = 5.678;

        a = a + e;

        System.out.println(e + " " + a);
    }

}
```

Whereas we always have to declare the type of variables in Java or Objective-C, we don't have to in Swift. The compiler calculates the type through type inference. Like in the example 'a' is implicit of type Double. Without the Int keyword, 'e' would have been implicit of type Int, because we assign 'e' with an Int value.

Multiple return values and tuples

Swift

```swift
func calculateMinMaxSum(a: Int, b: Int) -> (min: Int, max: Int, sum: Int) {

    if a > b {
        return (b, a, a + b)
    } else {
        return (a, b, a + b)
    }
}

let statistics = calculateMinMaxSum(5, b: 19)
let (min2, max2, sum2) = calculateMinMaxSum(5, b: 19)
print(sum2)             // prints 24
print(statistics.sum)   // prints 24
print(statistics.2)     // prints 24
```

There are no real equivalents to tuples or multiple return values in Java. You would have to return an extra type or array in Java, but often you would cause an overhead to create an extra class for these few properties and in an array the elements must be of the same type.

Strings

Swift

```swift
let dogLabel = "?"
let catLabel = "?"
let dogCount = 1
let catCount = 3
let wholeStr = "Hello, I have " + String(dogCount) + " " + dogLabel
               + " and \(catCount) " + catLabel + "s";
print(wholeStr) // Hello, I have 1 ? and 3 ?s
```

Java

```java
public class HelloWorld {

    public static void main(String[] args) {
        final String dogLabel = "\uD83D\uDE38";
        final String catLabel = "\uD83D\uDC36";
        final int dogCount = 1;
        final int catCount = 3;

        // usual approach
        final String wholeStr = "Hello, I have " + dogCount + " " + dogLabel
                        + " and " + catCount + " " + catLabel + "s";

        // using string interpolation
        // produces the same result
        final String wholeStrInterpol = "Hello, I have " + dogCount + " " + dogLabel
                        + " and " + String.format("%d", catCount) + " " + catLabel + "s";

        System.out.println(wholeStr);
    }

}
```

Class, Structs and Protocols

Swift

```swift
// Class
class SomeClass {

    var favLang: String

    init(favLang: String) {
        self.favLang = favLang
    }

}

let aClass = SomeClass(favLang: "Assembler")
let bClass = aClass
bClass.favLang = "Swift"

print(aClass.favLang) // "Swift"
print(bClass.favLang) // "Swift"

// Struct
struct SomeStruct {

    var favLang: String

    init(favLang: String) {
        self.favLang = favLang
    }

}

let aStruct = SomeStruct(favLang: "Assembler")
let bStruct = aStruct
bStruct.favLang = "Swift"

print(aStruct.favLang) // "Assembler"
print(bStruct.favLang) // "Swift"
```

Java

```java
public class SomeClass {

    public String favLang;

    public SomeClass(String favLang) {
        this.favLang = favLang;
    }

}
```

```java
public class Program {

    public static void main(String[] args) {

        final SomeClass aClass = new SomeClass("Assembler");
        final SomeClass bClass = aClass;

        aClass.favLang = "Swift";

        System.out.println(aClass.favLang); // "Swift"
        System.out.println(bClass.favLang); // "Swift"
    }

}
```

The Swift struct's example prints at first "Assembler" and then "Swift", because 'bStruct' doesn't hold a pointer to the same memory address, all values are copied and with the

statement “aClass.favLang = “Swift”” only the value of ‘aStruct’ changes. Consequently, structs are value types.

Switches and Patterns

Swift

```swift
let somePoint = (1, 1)

switch somePoint {
case (0, 0):
    print("origin")
case (let x, 0):
    print("I can do sth with the x")
case (-2...2, -2..<6):
    print("inside the box")
case let (x, y) where x == -y % 3:
    print("some calculations")
default:
    print("nirvana")
}
// prints "inside the box"
```

Java

```java
public class PointProgram {

    private static class Point {
        public final int x;
        public final int y;

        Point(int x, int y) {
            this.x = x;
            this.y = y;
        }

    }

    public static void main(String[] args) {

        Point p = new Point(1, 1); //The tuple '(1, 1)' from the Swift example was realize

        if (p.x == 0 && p.y == 0)
            System.out.println("origin");
        else if (p.y == 0) {
            final int y = p.y;
            System.out.println("I can do sth with the x");
        } else if (p.x >= -2 && p.x <= 2
                    && p.y >= -2 && p.y < 6)
            System.out.println("inside the box");
        else if (p.x == -p.y % 3) {
            System.out.println("some calculations");
        } else {
            System.out.println("nirvana");
        }

    }

}
```

There is no real tuple equivalent in Java, but to demonstrate the clarity in the Swift’s example, the tuple was realized with a private class.

Paradigms:

Swift	Java
Swift it is a multi-paradigm language that means we can do both functional and object-oriented programming with it. For example we can do the functional by using enums, and the object-oriented by using inheritance.	Java is a purely object-oriented paradigm that use the data (object) as a key element. This paradigm is an extension of imperative paradigm, it relies on sequential execution with a changing set of memory location and object implementation.

Dependencies:

	Swift	Java
Similarity	They are high level languages, the source code is translated to executable machine code by utilities such as compiler, Interpreter and linker, then translated into machine code.	
Differences	• It organizes code into modules, that code can be reused in other situations. • Initializer-based • Property-based • Parameter-based • Can handle the dependencies with package manager.	• It organizes code into packages, that code can be reused in other situations. • Class-based • Interface-based • Method-based • Can handle the dependencies with gradle build.

Memory Management:

Swift	Java
Swift uses the Automatic Reference Counting for memory management. ARC allocates a chunk of memory to store information about that instance. It provides a reference counting of the objects for the programming languages that works by counting the number of references to each object. When the reference count drops to zero the object is deallocated. Developers don't need to allocate or deallocate the objects explicitly.	Java uses an automatic memory management system called Garbage Collector for memory management. Applications will allocate additional memory to the heap area that is managed by the JVM. They keep track when the objects are no longer referenced in the code. GC can clean up the object and all related data will be removed automatically. Developers don't need to free or destroy the objects explicitly.

Part 2

User Login:

Swift :

```swift
    @IBAction func LogInTapped(_ sender: Any) {
        let email = EmailTextField.text!.trimmingCharacters(in: .whitespacesAndNewlines)
              let password = PasswordTextField.text!.trimmingCharacters(in: .whitespacesAndNewlines)
              // Signing in the user
              Auth.auth().signIn(withEmail: email, password: password) { (result, error) in

                  if error != nil {
                      // Couldn't sign in
                      self.ErrorLabel.text = error!.localizedDescription
                      self.ErrorLabel.alpha = 1
                  }
                  else {

                   if email.contains("admin@hotmail.com") {
                       let admainscreen = self.storyboard?.instantiateViewController(identifier:
                           Constants.Storyboard.adminHomeViewController) as? AdminHomeViewController

                       self.view.window?.rootViewController = admainscreen
                       self.view.window?.makeKeyAndVisible()

                   }else {
                       let accViewController = self.storyboard?.instantiateViewController(identifier:
                           Constants.Storyboard.accountViewController) as? AccountViewController

                       self.view.window?.rootViewController = accViewController
                       self.view.window?.makeKeyAndVisible()
                   }
                  }
              }
    }

}
```

Java:

```java
    public void logIn(View view) {

        String  email=email1.getText().toString();
        String  pass=pass1.getText().toString();

        if(email.isEmpty()|| pass.isEmpty()){

            message( titel: "Error", message: "Please fill all value",error);
        }
        else{
            Cursor c=db.signIn(email,pass);

            if(c.getCount() > 0) {

                message( titel: "Success",  message: "Logged IN", ok);
               // if admin account
                if(email1.getText().toString().equals("admin@hotmail.com")){
                    Intent intent = new Intent( packageContext: this, AdminAccount.class);
                    intent.putExtra( name: "email", email);
                    startActivity(intent);
                } else {
                    Intent intent = new Intent( packageContext: this, account.class);
                    intent.putExtra( name: "email", email);
                    startActivity(intent);
                }
            }
            else{
                message( titel: "Error", message: "Please check the email or password",error);
            }
        }
    }
```

The difference between Swift and Java login user feature, starting by declaring the function for the (sign in) button. In swift, the function does not need a returned type however java must declare the return type.

In swift we used *Authentication* function by passing the username and password to access firebase to reach the users in database. but in java we must declare *Cursor* object. Also, in Java we need to make a for loop to check every user in the database if it has a similar email and password entered by the user. If it is not an error message appears.

In Swift we use *view controller instantiate* to move to another page by declaring an *identifier* as the view controller ID. But in java we use *intent* to move to another page and using *putExtra* to have the information from this page to another.

User Registration:

Swift:

```
@IBAction func SignUpTapped(_ sender: Any) {

    let error = validateFields()
    if error != nil {
        showError(error!)
    }
    else {

        let Name = NameTextField.text!.trimmingCharacters(in: .whitespacesAndNewlines)
        let userName = UserNameTextField.text!.trimmingCharacters(in: .whitespacesAndNewlines)
        let email = EmailTextField.text!.trimmingCharacters(in: .whitespacesAndNewlines)
        let password = PasswordTextField.text!.trimmingCharacters(in: .whitespacesAndNewlines)
        let date = DueDateTextField.text!.trimmingCharacters(in: .whitespacesAndNewlines)
        Auth.auth().createUser(withEmail: email, password: password) { (result, err) in

            if err != nil {
                self.transitionToHome()
            }
            else {
                let db = Firestore.firestore()
                db.collection("users").addDocument(data: ["UserName":userName, "Email":email, "Pass":password,
                    "Name":Name, "Date":date ]) { (error) in

                    if error != nil {
                        self.transitionToHome()
                    }
                }
                self.transitionToHome()
            }
        }
    }
}
```

Java:

```
public void signUp(View view) {

    String user=username1.getText().toString();
    String email=email1.getText().toString();
    String name=name1.getText().toString();
    String date=date1.getText().toString();
    String pass=pass1.getText().toString();
    String conpass=conpass1.getText().toString();
    if (user.isEmpty()||email.isEmpty()||name.isEmpty()||date.isEmpty()||pass.isEmpty()||conpass.isEmpty()){
        message( titel: "Error", message: "Please fill all value",error);
    }
    else{
        if(pass.equals(conpass)){

            boolean  insert=db.insertdataU(sqLiteDatabase,user,pass,name,email,date);

            if(insert==true){
                Intent intent = new Intent( packageContext: this, SignIn.class);
                startActivity(intent);

                clearText();}
            else{
                message( titel: "Error",  message: "Record not added",error);
            }
        }
        else{
            message( titel: "Error", message: "password and confirm password not compatible",error);
        }
    }

}
```

To add a new user to the firebase in swift we used *firestore* function to connect to the firestore first. Then we used *addDocument* function to add the user by passing all the information to the database.

But in java we used the function *insertData* by passing the *SQLiteDatabase* object and the required user information as parameters. *insertData* function was already declared.

```
public boolean insertdataU(SQLiteDatabase db, String password, String uname, String Username, String Uemaill, String Date) {

    String sql = "INSERT INTO user (username,pass,name,emaill,date) VALUES (?,?,?,?,?)";
    SQLiteStatement statement = db.compileStatement(sql);
    statement.bindString( index: 1, Username);
    statement.bindString( index: 2, password);
    statement.bindString( index: 3, uname);
    statement.bindString( index: 4, Uemaill);
    statement.bindString( index: 5, Date);
    statement.execute();

        return true;

}
```

Frequently Asked Questions (FAQ) Page (User):

When we talk about the difference between Swift and Java in this page, there are two main things: first how the FAQ list is structured, and second the functions in this page. For the FAQ list was structured in Swift using *UITableView* which is the most commonly used by apps whose data is highly structured. It displays a single column of vertically scrolling content, divided into rows. Each row in the table will contain question and answer. In order to customize the content of each row to a custom style based on our app needs, a *UITableViewCell* objects was used to have a unified appearance in all rows.

On the other hand, the FAQ list was structured in Java using *ListView* where it displays a vertically scrollable collection of items, where each item in the list will contain question and answer. In order to customize the content of each item to a custom style based on our app needs, a custom *XML layout* was used to have a unified appearance in all rows.

The two figures below show the difference between Swift and Java in the custom cell structure. For swift, the custom cell is implemented in the same interface and for java, the custom cell is implemented different interface.

Swift:

Java:

We used different techniques to map the above custom cell and implement it in every row in the list. When the interface starts running, for swift, the system will retrieve FAQ from cloud firestore database in array of objects of type Community and use two protocols *UITableViewDataSource* protocol and *UITableViewDelegate* protocol.

UITableViewDelegate protocol was used to manage selections and reordering of cells and the number of sections in the table view whereas *UITableViewDataSource* protocol was used to manage the data itself and provide cells for a table view as shown in the Figure below.

On the other hand, for java, the system will retrieve FAQ from SQLite database into two arrays of type String and pass it to the adapter class *communityListAdapter* that shown in the figure below. This class is used to manage the data itself and to fill the ListView with the data.

Swift:

```
import UIKit
import FirebaseFirestore
import Firebase

struct Community {
    var Question: String
    var Answer: String
    var documentId: String

}

class FAQviewController: UIViewController, UITableViewDataSource, UITableViewDelegate {

    func numberOfSections(in tableView: UITableView) -> Int {
           return 1
       }


       func tableView(_ tableView: UITableView, numberOfRowsInSection section: Int) -> Int {
           return self.CommunityArrey.count
       }

       func tableView(_ tableView: UITableView, cellForRowAt indexPath: IndexPath) -> UITableViewCell {

           let cell = tableView.dequeueReusableCell(withIdentifier: "cell", for:indexPath) as! TableViewCell
           cell.configureCell(faq: CommunityArrey[indexPath.row])
           return cell

       }
import UIKit

class TableViewCell: UITableViewCell {
    @IBOutlet weak var questionLabel: UILabel!
    @IBOutlet weak var answerLabel: UILabel!

    override func awakeFromNib() {
        super.awakeFromNib()
    }

    func configureCell(faq: Community){
        questionLabel.text = faq.Question
        answerLabel.text = faq.Answer
    }
}
```

Java:

```
public class communityListAdapter extends BaseAdapter{
    Context context;
    ArrayList<String> question1;
    ArrayList<String> answer1;
    public communityListAdapter(Context context2, ArrayList<String> question1, ArrayList<String> answer1) {
        this.context = context2;
        this.question1 = question1;
        this.answer1 = answer1;
    }
    public int getCount() { return answer1.size(); }
    public Object getItem(int position) { return null; }
    public long getItemId(int position) { return 0; }
    public View getView(int position, View child, ViewGroup parent) {
        Holder holder;
        LayoutInflater layoutInflater;
        if (child == null) {
            layoutInflater = (LayoutInflater) context.getSystemService(Context.LAYOUT_INFLATER_SERVICE);
            child = layoutInflater.inflate(R.layout.single_question_answer,  root: null);
            holder = new Holder();
            holder.Question_TextView = child.findViewById(R.id.babySize);
            holder.Answer_TextView = child.findViewById(R.id.newFacts);
            child.setTag(holder);

        } else {
            holder = (Holder) child.getTag();
        }
        holder.Question_TextView.setText(question1.get(position));
        holder.Answer_TextView.setText(answer1.get(position));
        return child;
    }
    public class Holder {
        TextView Question_TextView;
        TextView Answer_TextView;
    }

}
```

Also, in order to load the data from the database, for swift, we used a user defined function called *loadData()* that is shown in the figure below. This function gets the needed data from firebase cloud firestore as *querySnapshot* and then loops through that snapshot to get all the questions and answers and appends it to the object array *CommunityArrey* in order to fill it in the list.

On the contrary, for java, we use a user defiend function called *ShowSQLiteDBdata()* that is shown in the figure below. This function saves a cursor of the result set returned by a database query on table Community then loops through that cursor to get all the questions and answers and append it to a string array in order to fill it in the list.

Swift:

```swift
//variables
var CommunityArrey = [Community]()
var db:Firestore!

override func viewDidLoad() {
    super.viewDidLoad()
    db = Firestore.firestore()
    tableView.delegate = self
    tableView.dataSource = self
    tableView.autoresizingMask = UIView.AutoresizingMask.flexibleHeight
    setUpElements()
    loadData()

}
func loadData(){
       db.collection("Community").getDocuments { (querySnapshot, error) in
           if let error = error
           {
               print("Error getting documents: \(error)")
           }
           else
           {
               for document in querySnapshot!.documents {
                   let data = document.data()
                   let theQue = data["Question"] as! String
                   let theAns = data["Answer"] as! String
                   let documentId = document.documentID

                   let newFAQ = Community(Question: theQue, Answer: theAns, documentId: documentId)
                   self.CommunityArrey.append(newFAQ)

               }
               self.tableView.reloadData()
           }
```

Java:

```java
    DatabaseHelper sqLiteHelper;
    SQLiteDatabase sqLiteDatabase;
    Cursor cursor;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.q_a_community);
        sqLiteHelper = new DatabaseHelper( context: this);
        question = new ArrayList<>();
        answer = new ArrayList<>();
        LISTVIEW = findViewById(R.id.listMenu);
        questionArea = findViewById(R.id.editText);
        ask = findViewById(R.id.button);
}
    @Override
    protected void onResume() {...}

    private void ShowSQLiteDBdata() {
        sqLiteDatabase = sqLiteHelper.getWritableDatabase();
        cursor = sqLiteDatabase.query(DatabaseHelper.COMMUNITY_TABLE, columns: null, selection: null,  selectionArgs: null,
        question.clear();
        answer.clear();
        cursor.moveToFirst();
        if (cursor != null && cursor.moveToFirst()) {
            do {
                question.add(cursor.getString(cursor.getColumnIndex(cursor.getColumnName( columnIndex: 0))));
                answer.add(cursor.getString(cursor.getColumnIndex(cursor.getColumnName( columnIndex: 1))));

            } while (cursor.moveToNext());
        }
        listAdapter = new communityListAdapter( context2: QA_Community.this,question,answer);
        LISTVIEW.setAdapter(listAdapter);
        cursor.close();
    }
```

FAQ Page (User) – Ask a Question:

The user can ask a question if it has not been asked before. When the user asks a question, it should be saved in the database in order to answer it by the admin. For swift, we used a user defined function called *saveQuestion()* that adds the asked question to the

firestore database using a predefined function addDocument(data:[]). It requires only one line to add data to database.

On the other hand, for java, we use a user defined function called *askQuestion()* that adds the asked question to the SQLite database using another user defined function from the database class called *insertQuestion(question, db).* It requires around 4 lines to add data to database.

Swift:

```
@IBAction func saveQuestion(_ sender: Any) {
    let error = validateFields()
    if error != nil {
        showError(error!)
    }
    else {
        let newQues = getQuestion.text!.trimmingCharacters(in: .whitespacesAndNewlines)
        db.collection("AskedQuestions").addDocument(data: ["Question" : newQues])
        let message = "your question will be answerd within 1 day, thank you."
        let alertController:UIAlertController = UIAlertController(title: "Message", message: message, preferredStyle:
            UIAlertController.Style.alert)
        let alertAction:UIAlertAction = UIAlertAction(title: "OK", style: UIAlertAction.Style.default, handler:nil)
        alertController.addAction(alertAction)
        self.present(alertController, animated: true, completion: nil)
    }
}
```

Java:

```
public void askQuestion(View view) {
    String questionAsked = questionArea.getText().toString().trim();

    if (questionAsked.length() > 0) {
        DatabaseHelper.insertQuestion( question: questionAsked + "?", sqLiteDatabase);
        showMessage( title: "success", message: "your question will be answerd within 1 day, thank you." );
        questionArea.setText("");
    }
    else if(questionAsked.length() == 0){
        Toast.makeText( context: QA_Community.this, text: "please ask a question before click the button.",Toast.LENGTH_LONG).show();
    }
}
```

```
public static void insertQuestion(String question, SQLiteDatabase db) {

    String sql = "INSERT INTO Questions (question) VALUES (?)";
    SQLiteStatement statement = db.compileStatement(sql);
    statement.bindString( index: 1, question);
    statement.execute();
}
```

Frequently Asked Questions (FAQ) Page (Admin):

For the structure of the FAQ list, it is the same as the *(Frequently Asked Questions (FAQ) Page (user*)) that was mentioned previously. The only difference here is that the admin can select a question from the list to answer it. When the admin selects a question, an empty label will be filled by that question. The two figures below show the difference between swift and java when handling the select. For swift, it uses the instance method

tableView(:didSelectRowAt:) that tells the table delegate that the specified row is now selected. And for java, we will set *OnItemClickListener()* that implement *onItemClick()* method which tells the listView that there is an item being selected.

Swift:

```
func tableView(_ tableView: UITableView, didSelectRowAt indexPath: IndexPath) {
    let theQuestionSelected = AskedQuestionsArrey[indexPath.row].Question
    theQuestionSelectedID = AskedQuestionsArrey[indexPath.row].documentId
    self.questionTextView.text = theQuestionSelected

}
```

Java:

```
LISTVIEW.setOnItemClickListener(new AdapterView.OnItemClickListener() {
    @Override
    public void onItemClick(AdapterView<?> adapterView, View view, int position, long id) {
        quest = question.get(position).trim();
        Ques.setText(quest);
    }
});
```

FAQ Page (Admin) – Answer a Question:

For answering a question, it is mostly the same as the *(Frequently Asked Questions (FAQ) Page (user*)) that was mentioned previously when it comes to the system validation and database insertion. The only difference here is that when the admin chooses a question from the list and answers it, that question will be removed from the database as well as from the list to avoid answering the same question repeatedly. For swift, it requires only one line to delete document from firestore database whereas in java, we use a user defined function from the database class called *deleteAnswerdQuestion (question, db).*

Swift:

```swift
    @IBAction func saveAnsweredQuestion(_ sender: Any) {
        var message = ""
        let newQues = questionTextView.text!.trimmingCharacters(in: .whitespacesAndNewlines)
        if newQues == "Question"{
            message = "Please choose a question before click the button."
        }
        else{
            if answerTextFeild.text?.trimmingCharacters(in: .whitespacesAndNewlines) == "" {
                message = "Please answer the question before click the button."
            }
            else{
                let newAns = answerTextFeild.text!.trimmingCharacters(in: .whitespacesAndNewlines)

                db.collection("Community").addDocument(data: ["Question" : newQues, "Answer": newAns])
                db.collection("AskedQuestions").document(theQuestionSelectedID).delete()
                self.questionTextView.text = "Question"
                self.answerTextFeild.text = ""
                loadData()

                message = "your answer will be added to the questions community, thank you."
            }
        }
        let alertController:UIAlertController = UIAlertController(title: "Message", message: message, preferredStyle:
            UIAlertController.Style.alert)
        let alertAction:UIAlertAction = UIAlertAction(title: "OK", style: UIAlertAction.Style.default, handler:nil)
        alertController.addAction(alertAction)
        self.present(alertController, animated: true, completion: nil)

    }
```

Java:

```java
public void answerQuestion(View view) {
    if (Ques.getText().toString().trim().equals("question") || Ques.getText().toString().trim().equals("")) {
        Toast.makeText( context: QA_CommunityAdmin.this,  text: "please choose a question before click the button.", Toast.LENGTH_LONG
    } else {
        String questionAnswered = answerArea.getText().toString().trim();

        if (questionAnswered.length() > 0) {
            DatabaseHelper.insertAnswerdQuestion(quest, questionAnswered, sqLiteDatabase);
            DatabaseHelper.deleteAnswerdQuestion(quest, sqLiteDatabase);
            showMessage( title: "success",  message: "your answer will be added to the questions community, thank you.");
            ShowSQLiteDBdata();
            answerArea.setText("");
            quest = "";
        } else if (questionAnswered.length() == 0) {
            Toast.makeText( context: QA_CommunityAdmin.this,  text: "please answer the question before click the button.", Toast.LENGT
        }
    }
}
```

```java
    public static void deleteAnswerdQuestion(String quest, SQLiteDatabase db) {
        String deleteQuestion = "delete from Questions where question = '"+ quest +"'";
        db.execSQL(deleteQuestion);
    }
```

Home:

Swift:

```swift
class HomeViewController: UIViewController {

    @IBOutlet weak var monthNumber: UIImageView!

    @IBOutlet weak var monthImage: UIImageView!

    @IBOutlet weak var monthDesc: UILabel!

    @IBOutlet weak var monthSize: UIImageView!

    @IBOutlet weak var monthNewDesc: UILabel!

    @IBOutlet weak var Home: UIButton!

    @IBOutlet weak var community: UIButton!

    @IBOutlet weak var appointment: UIButton!

    @IBOutlet weak var account: UIButton!

    var userDate = ""

    var month1Desc = "By the end of this month, the baby is the size of a grain of rice!"

    var month2Desc = "By the end of this month, the baby is the size of a cherry!"

    var month3Desc = "By the end of this month, the baby is the size of a lemon!"

    var month4Desc = "By the end of this month, the baby is the size of a pear!"

    var month5Desc = "By the end of this month, the baby is the size of a grapefruit!"

    var month6Desc = "By the end of this month, the baby is the size of an eggplant!"

    var month7Desc = "By the end of this month, the baby is the size of a pineapple!"

    var month8Desc = "By the end of this month, the baby is the size of a papaya!"

    var month9Desc = "By the end of this month, the baby is the size of a watermelon!"

    var m1NewDesc = "Your baby is only .1 to .2 millimeters and at this stage is called a blastocyst. At three weeks pregnant, your child has already developed all his genetic material — and the sex is already decided."

    var m2NewDesc = "Your baby is now a little under an inch long but has developed into a tiny human being. The heart is beating, the brain is developing and she has developed all her limbs as well as hands and feet."

    var m3NewDesc = "Your baby is now officially a fetus and is between two and four inches long. By the end of the first trimester, all organs are present, and even fingernails are developing. Your baby is also moving her arms and legs."

    var m4NewDesc = "Your baby is five to six inches long and weighs up to four ounces. Baby's face and heart are fully formed at this point, though the lungs are still developing. Baby's eyes will open during this month."

    var m5NewDesc = "Your baby is now about 10 ounces and six to nine inches long. Baby is covered with a fine protective hair. This month he or she will develop fingerprints and permanent teeth buds behind fully formed baby teeth. Sex can be determined."

    var m6NewDesc = "Your baby is about 10 inches long and weighs over a pound. You'll be aware of baby's movements as he or she stretches and hiccups. Baby's eyes now open and close, vocal cords are functioning."

    var m7NewDesc = "Your baby is starting to develop fat under his or her skin. Baby's now almost 12 inches long and weighs between two and four pounds. Your child can now see, hear and taste, and the brain and nervous system are growing rapidly."

    var m8NewDesc = "Your baby's brainwaves resemble those of a newborn by this time. He may be a foot or longer and weigh about five pounds.  The lungs and brain are continuing to develop and other body systems are refining to be ready for living outside."

    var m9NewDesc = "Baby's lungs are maturing, and he/she is shedding the layer of hair that protected him/her in the uterus. Your baby's brain is growing tremendously this last month. Baby measures about 18 to 21 inches long and weighs about 6 to 8 pounds."

    func setUpElements(){

        Utilities.styleNavigButton(Home)
        Utilities.styleNavigButton(community)
        Utilities.styleNavigButton(appointment)
        Utilities.styleNavigButton(account)

    }
```

```
80      override func viewDidLoad() {
81          super.viewDidLoad()
82          // Do any additional setup after loading the view.
83
84           setUpElements()
85
86          let db = Firestore.firestore()
87
88          let user = Auth.auth().currentUser
89
90          if let user = user {
91
92              let uid = user.uid          △ Initialization of immutable value 'uid' was never used; consider replacing with assignment to '_' or removing it
93
94              let docRef = db.collection("users").document("ugPSaXUFOfbm2kk9IzYc")
95
96          docRef.getDocument { (document, error) in
97              if let document = document, document.exists {
98                  let dataDescription = document.data().map(String.init(describing:)) ?? "nil"
99                  print("Document data: \(dataDescription)")
100
101                 let data = document.data()
102
103                 self.userDate = data?["Date"] as! String
104
105                 print(self.userDate)
106
107                 var currentMonth = Int(self.userDate)          △ Variable 'currentMonth' was never mutated; consider changing to 'let' constant
108                 print(currentMonth)                            △ Expression implicitly coerced from 'Int?' to 'Any'
109
110 //                self.monthNumber.image = UIImage(named: "month\(currentMonth)")
111 //                self.monthImage.image = UIImage(named: "m\(currentMonth)")
112
113                 if currentMonth == 1 {
114
115                     self.monthDesc.text =  self.month1Desc
116                     self.monthSize.image = UIImage(named: "wheat")
117                     self.monthNewDesc.text =  self.m1NewDesc
118
119
120                   }
121
```

```
122                   else if currentMonth == 2 {
123
124                    self.monthDesc.text =  self.month2Desc
125                    self.monthSize.image = UIImage(named: "cherry")
126                    self.monthNewDesc.text =  self.m2NewDesc
127
128                   }
129
130                   else if currentMonth == 3 {
131
132                    self.monthDesc.text =  self.month3Desc
133                    self.monthSize.image = UIImage(named: "lemon")
134                    self.monthNewDesc.text =  self.m3NewDesc
135
136                   }
137
138                   else if currentMonth == 4 {
139
140                    self.monthDesc.text =  self.month4Desc
141                    self.monthSize.image = UIImage(named: "pear")
142                    self.monthNewDesc.text =  self.m4NewDesc
143
144                   }
145
146                   else if currentMonth == 5 {
147
148                    self.monthDesc.text =  self.month5Desc
149                    self.monthSize.image = UIImage(named: "grapefruit")
150                    self.monthNewDesc.text =  self.m5NewDesc
151
152                   }
153
154                   else if currentMonth == 6 {
155
156                    self.monthDesc.text =  self.month6Desc
157                    self.monthSize.image = UIImage(named: "eggplant")
158                    self.monthNewDesc.text =  self.m6NewDesc
159
160                   }
161
```

```
                    else if currentMonth == 7 {

                     self.monthDesc.text =  self.month7Desc
                     self.monthSize.image = UIImage(named: "pineapple")
                     self.monthNewDesc.text =  self.m7NewDesc

                    }

                    else if currentMonth == 8 {

                     self.monthDesc.text =  self.month8Desc
                     self.monthSize.image = UIImage(named: "papaya")
                     self.monthNewDesc.text =  self.m8NewDesc
                     self.monthImage.image = UIImage(named: "m8")
                     self.monthNumber.image = UIImage(named: "month8")

                    }

                    else if currentMonth == 9 {

                     self.monthDesc.text =  self.month9Desc
                     self.monthSize.image = UIImage(named: "watermelon")
                     self.monthNewDesc.text =  self.m9NewDesc

                    }



            } else {
                print("Document does not exist")
            }
        }


}


}
 }
```

Java:

```
public class home extends AppCompatActivity {

    DatabaseHelper helper ;

    int currentMonth = 0;

    String date;

    ImageView monthTitle;
    ImageView monthPicture;
    TextView babySizeDescription;
    ImageView babySizePicture;
    TextView newFactsDescription;
    Button home, community, appointments, account;
    String email;

    @RequiresApi(api = Build.VERSION_CODES.O)
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_home);

        helper = new DatabaseHelper( context: this);

        Intent i = getIntent();
        date = i.getStringExtra( name: "due");

        currentMonth = helper.cutMonth(date);


        System.out.println("date : " + date + "current : " + currentMonth);
```

```
    monthTitle = findViewById(R.id.currentMonth);
    monthPicture = findViewById(R.id.monthPic);
    babySizeDescription = findViewById(R.id.babySizeDesc);
    babySizePicture = findViewById(R.id.babySizePic);
    newFactsDescription = findViewById(R.id.newFactsDesc);
    home=(Button)findViewById(R.id.home);
    community=(Button)findViewById(R.id.community);
    appointments=(Button)findViewById(R.id.appointment);
    account=(Button)findViewById(R.id.account);
    Intent intent= getIntent();
    email = intent.getStringExtra( name: "email");

    String m1babySize = "By the end of this month, the baby is the size of a grain of rice!";
    String m2babySize = "By the end of this month, the baby is the size of a cherry!";
    String m3babySize = "By the end of this month, the baby is the size of a lemon!";
    String m4babySize = "By the end of this month, the baby is the size of a pear!";
    String m5babySize = "By the end of this month, the baby is the size of a grapefruit!";
    String m6babySize = "By the end of this month, the baby is the size of an eggplant!";
    String m7babySize = "By the end of this month, the baby is the size of a pineapple!";
    String m8babySize = "By the end of this month, the baby is the size of a papaya!";
    String m9babySize = "By the end of this month, the baby is the size of a watermelon!";

    String m1facts = "Your baby is only .1 to .2 millimeters and at this stage is called a blastocyst. At three weeks pregnant, your child has already
    String m2facts = "Your baby is now a little under an inch long but has developed into a tiny human being. The heart is beating, the brain is devel
    String m3facts = "Your baby is now officially a fetus and is between two and four inches long. By the end of the first trimester, all organs are p
    String m4facts = "Your baby is five to six inches long and weighs up to four ounces. Baby's face and heart are fully formed at this point, though
    String m5facts = "Your baby is now about 10 ounces and six to nine inches long. Baby is covered with a fine protective hair. This month he or she
    String m6facts = "Your baby is about 10 inches long and weighs over a pound. You'll be aware of baby's movements as he or she stretches and hiccup
    String m7facts = "Your baby is starting to develop fat under his or her skin. Baby's now almost 12 inches long and weighs between two and four pou
    String m8facts = "Your baby is starting to develop fat under his or her skin. Baby's now almost 12 inches long and weighs between two and four pou
    String m9facts = "Baby's lungs are maturing, and he or she is shedding the layer of hair that protected him or her in the uterus. Your baby's brai
```

```
    if(currentMonth == 1) {

        monthTitle.setImageResource(R.drawable.month1);
        monthPicture.setImageResource(R.drawable.m1);
        babySizeDescription.setText(m1babySize);
        babySizePicture.setImageResource(R.drawable.wheat);
        newFactsDescription.setText(m1facts);

    }

    else if(currentMonth == 2) {

        monthTitle.setImageResource(R.drawable.month2);
        monthPicture.setImageResource(R.drawable.m2);
        babySizeDescription.setText(m2babySize);
        babySizePicture.setImageResource(R.drawable.cherry);
        newFactsDescription.setText(m2facts);

    }

    else if(currentMonth == 3) {

        monthTitle.setImageResource(R.drawable.month3);
        monthPicture.setImageResource(R.drawable.m3);
        babySizeDescription.setText(m3babySize);
        babySizePicture.setImageResource(R.drawable.lemon);
        newFactsDescription.setText(m3facts);

    }

    else if(currentMonth == 4) {

        monthTitle.setImageResource(R.drawable.month4);
        monthPicture.setImageResource(R.drawable.m4);
        babySizeDescription.setText(m4babySize);
        babySizePicture.setImageResource(R.drawable.pear);
        newFactsDescription.setText(m4facts);
```

```
    else if(currentMonth == 5) {

        monthTitle.setImageResource(R.drawable.month5);
        monthPicture.setImageResource(R.drawable.m5);
        babySizeDescription.setText(m5babySize);
        babySizePicture.setImageResource(R.drawable.grapefruit);
        newFactsDescription.setText(m5facts);

    }

    else if(currentMonth == 6) {

        monthTitle.setImageResource(R.drawable.month6);
        monthPicture.setImageResource(R.drawable.m6);
        babySizeDescription.setText(m6babySize);
        babySizePicture.setImageResource(R.drawable.eggplant);
        newFactsDescription.setText(m6facts);

    }

    else if(currentMonth == 7) {

        monthTitle.setImageResource(R.drawable.month7);
        monthPicture.setImageResource(R.drawable.m7);
        babySizeDescription.setText(m7babySize);
        babySizePicture.setImageResource(R.drawable.pineapple);
        newFactsDescription.setText(m7facts);

    }

    else if(currentMonth == 8) {

        monthTitle.setImageResource(R.drawable.month8);
        monthPicture.setImageResource(R.drawable.m8);
        babySizeDescription.setText(m8babySize);
        babySizePicture.setImageResource(R.drawable.papaya);
        newFactsDescription.setText(m8facts);

    }
```

```
        else if(currentMonth == 9) {

            monthTitle.setImageResource(R.drawable.month9);
            monthPicture.setImageResource(R.drawable.m9);
            babySizeDescription.setText(m9babySize);
            babySizePicture.setImageResource(R.drawable.watermelon);
            newFactsDescription.setText(m9facts);

        }

        home.setOnClickListener((v) -> {
                Intent i=new Intent(getApplicationContext(), home.class);
                i.putExtra( name: "email", email);
                startActivity(i);
        });

        community.setOnClickListener((v) -> {
                Intent i=new Intent(getApplicationContext(), QA_Community.class);
                i.putExtra( name: "email", email);
                startActivity(i);
        });
        appointments.setOnClickListener((v) -> {
                Intent i=new Intent(getApplicationContext(), appointment.class);
                i.putExtra( name: "email", email);
                startActivity(i);
        });
        account.setOnClickListener((v) -> {
                Intent i=new Intent(getApplicationContext(), account.class);
                i.putExtra( name: "email", email);
                startActivity(i);
        });

    }
}
```

```
    @RequiresApi(api = Build.VERSION_CODES.O)
    public int cutMonth(String s) {

        LocalDate currentdate = LocalDate.now();
        int currentMonth = currentdate.getMonthValue();
        int currentStr = Integer.parseInt(s.substring(s.indexOf("-") + 1, s.lastIndexOf( str: "-")));
        int result = currentStr - currentMonth;

        return result;


    }
```

In swift, the current month was saved in the firebase database, while in java it was saved in the SQL database. The way they were saved was quite different too. In swift, the current month was saved as an integer in Firebase, while in java it was saved as a full date and a function had to be written to convert it into an integer.

Another difference that is noticed is that in swift, the getDocument method is used to fetch the current month from the firebase database, however in java, an sql query was used to retrieve it.

Finally, the images and resources were set using the .text and .image methods in swift, whereas in java, the .setImageResource and .setText methods were used.

Account Page:

There are some differences in creating this page between Swift and Java, including:

1) TextView definition and creation method.
   - Swift : by `@IBOutlet weak var Name: UITextView!`
   - Java : by `TextView Name = findViewById(R.id.name);`

2) First function invocation.
   - Swift : `viewDidLoad()`
   - Java : `onCreate()`

3) Reverence to database.
   - Swift : `docRef = Firestore.firestore().collection("users")`
   - Java : `DatabaseHelper helper;`

4) Getting user information.
   - Swift : `docRef.getDocument`
   - Java : `helper.userProfilInfo(email);`

5) Seting information into TextView.
   - Swift :
     ```
     self.name = data?["Name"] as! String
     self.UserName.text = self.name
     ```
   - Java :
     ```
     String n = result.getString(2);
     name.setText(n);
     ```

Swift :

For Showing Admin Info:

```
class AdminHomeViewController: UIViewController {

    @IBOutlet weak var AdminName: UITextView!
    @IBOutlet weak var AdminType: UITextView!
    @IBOutlet weak var AdminPassword: UITextView!
    @IBOutlet weak var UpdateViewButton: UIButton!
    @IBOutlet weak var FAQButton: UIButton!
    var name = ""
    var password = ""
    var due = ""
    override func viewDidLoad() {
        super.viewDidLoad()
        viewprofile()
    }
    func viewprofile() {
        let db = Firestore.firestore()
        let user = Auth.auth().currentUser
        if let user = user {
            let uid = user.uid
            let docRef = db.collection("users").document(uid)
            docRef.getDocument { (document, error) in
                if let document = document, document.exists {
                    let dataDescription = document.data().map(String.init(describing:)) ??
                        "nil"
                    print("Cached document data: \(dataDescription)")
                    let data = document.data()
                    self.name = data?["Name"] as! String
                    self.password = data?["Pass"] as! String
                    self.AdminName.text = self.name
                    self.AdminType.text = "Admin"
                    self.AdminPassword.text = self.password
                } else {
                    print("Document does not exist")
                }
            }
        }
    }
    func transitionToEditeAdimnProfile() {
        let adminEditeProfileViewController =
            storyboard?.instantiateViewController(withIdentifier:
            Constants.Storyboard.adminEditeProfileViewController) as?
            EditAdminProfileViewController
        view.window?.rootViewController = adminEditeProfileViewController
        view.window?.makeKeyAndVisible()
    }
```

For Showing User Info:

```
class AccountViewController: UIViewController {
    @IBOutlet weak var UserName: UITextView!
    @IBOutlet weak var UserType: UITextView!
    @IBOutlet weak var UserPassword: UITextView!
    @IBOutlet weak var UserDueDate: UITextView!
    var name = ""
    var password = ""
    var due = ""

    override func viewDidLoad() {
        super.viewDidLoad()
        viewprofile()
    }
    func viewprofile() {
        let db = Firestore.firestore()
            let user = Auth.auth().currentUser
            if let user = user {
                let uid = user.uid
                let docRef = db.collection("users").document(uid)
                docRef.getDocument { (document, error) in
                if let document = document, document.exists {
                    let dataDescription = document.data().map(String.init(describing:)) ??
                        "nil"
                    print("Cached document data: \(dataDescription)")
                    let data = document.data()
                    self.name = data?["Name"] as! String
                    self.password = data?["Pass"] as! String
                    self.due = data?["Date"] as! String
                    self.UserName.text = self.name
                    self.UserType.text = "Mom"
                    self.UserPassword.text = self.password
                    self.UserDueDate.text = self.due
                    } else {
                        print("Document does not exist")
                    }
        }
        }
    }

    func transitionToEditeUserProfile() {
        let usereditеprofileViewController =
            storyboard?.instantiateViewController(withIdentifier:
            Constants.Storyboard.usereditеprofileViewController) as?
            EditUserProfileViewController
        view.window?.rootViewController = usereditеprofileViewController
        view.window?.makeKeyAndVisible()
    }
```

Java :

For Showing User Info:

```
public class account extends AppCompatActivity {

    TextView name;
    TextView due;
    TextView usertype;
    TextView pas;
    Bundle bundle;
    String email, d;
    Cursor result;
    DatabaseHelper helper ;


    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_account);
        Intent i = getIntent();
        email = i.getStringExtra( name: "email");
        helper = new DatabaseHelper( context: this);
        result=helper.userProfileInfo(email);
        name=(TextView) findViewById(R.id.name);
        usertype=(TextView) findViewById(R.id.userType);
        pas=(TextView) findViewById(R.id.pas);
        due=(TextView) findViewById(R.id.duedate);
        home=(Button)findViewById(R.id.home);
        community=(Button)findViewById(R.id.community);
        appointments=(Button)findViewById(R.id.appointment);
        account=(Button)findViewById(R.id.account);

        if (result.getCount() == 0) {
            Toast.makeText( context: this,  text: "No Match", Toast.LENGTH_LONG).show();
        } else {
            result.moveToFirst();
            String u = result.getString( i: 0);
            String p = result.getString( i: 1);
            String n = result.getString( i: 2);
             d = result.getString( i: 3);
            name.setText(n);
            due.setText(d);
            pas.setText(p);
            usertype.setText(u);
        }
    }
    public void UpdateInfo(View view) {
        Intent intent = new Intent( packageContext: this, editUser.class);
        intent.putExtra( name: "email", email);
        startActivity(intent);
    }
```

Helper :

```
    public Cursor userProfileInfo(String e) {
        SQLiteDatabase db = this.getWritableDatabase();
        return db.rawQuery( sql: "Select username,pass,name,date from '" + USER_TABLE + "' where " +
                "emaill= ? ",new String[]{e});
    }
```

For Showing Admin Info:

```
public class AdminAccount extends AppCompatActivity {

    TextView name;
    TextView usertype;
    TextView pas;
    Bundle bundle;
    String email;
    Cursor result;
    DatabaseHelper helper ;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_admin_account);
        Intent i = getIntent();
        bundle = i.getExtras();
        email = bundle.getString( key: "email");
        helper = new DatabaseHelper( context: this);
        result=helper.adminProfileInfo(email);
        name=(TextView) findViewById(R.id.nameAdmin);
        usertype=(TextView) findViewById(R.id.type);
        pas=(TextView) findViewById(R.id.pasA);
        if (result.getCount() == 0) {
            Toast.makeText( context: this,  text: "No Match", Toast.LENGTH_LONG).show();
        } else {
            result.moveToFirst();
            String u = result.getString( i: 0);
            String p = result.getString( i: 1);
            String n = result.getString( i: 2);
            name.setText(n);
            pas.setText(p);
            usertype.setText(u);
        }
    }
    public void UpdateInfo(View view) {
        Intent intent = new Intent( packageContext: this, editAdmin.class);
        intent.putExtra( name: "email", email);
        startActivity(intent);
    }
```

Helper :

```
    public Cursor adminProfileInfo(String e) {
        SQLiteDatabase db = this.getWritableDatabase();
        return db.rawQuery( sql: "Select username,pass,name from '" + USER_TABLE + "' where emaill" +
                "= ? ",new String[]{e});
    }
```

Edit Page:

There are some differences in creating this page between Swift and Java, including:

1) EditText definition and creation method.
    - Swift : by `@IBOutlet weak var EditedName: UITextField!`
    - Java : by `EditText Name = findViewById(R.id.name);`

2) Get user input.
    - Swift : `EditedName.text!.trimmingCharacters`
    - Java : `Name.getText().toString().trim();`

3) Update information on database.
    - Swift :
```
database.collection("users").document(id)
.updateData(["Name": name,])
```
    - Java : `helper.updateUserName(email , Name);`

4) Check some input.
    - Swift :
```
Utilities.isName(newName) == false {
return "Please make sure you enter a valid name."
```
    - Java : by `android:inputType="textPersonName"`

5) Show error.
    - Swift : `if error != nil {showError(error!) }`
    - Java : `Message("Error", "Error Massage");`

Java offers a feature to verify the input by showing a keyboard suitable for the input type, and this does not apply to Swift. In addition, Swift does not support SQLite database but Java supports Firebase database.

Swift :

For Editing Admin Info:

```
class EditAdminProfileViewController: UIViewController {
    @IBOutlet weak var EditAdminError: UILabel!
    @IBOutlet weak var AdminEditedName: UITextField!
    @IBOutlet weak var AdminEditedPassword: UITextField!
    @IBOutlet weak var UpdateAdminInfo: UIButton!
    @IBOutlet weak var BackButton: UIButton!
    override func viewDidLoad() {
        super.viewDidLoad()
        setUpElements()
    }
    func setUpElements (){
        EditAdminError.alpha = 0
    }
    @IBAction func EditAdminInfo(_ sender: Any) {
        let error = validateFields()
        if error != nil {
            showError(error!)
        }
        else {
            let db = Firestore.firestore()
            let user = Auth.auth().currentUser
            let id = user.uid
            let name = AdminEditedName.text!.trimmingCharacters(in: .whitespacesAndNewlines)
            let password = AdminEditedPassword.text!.trimmingCharacters(in:
                .whitespacesAndNewlines)
            let database = Firestore.firestore()
            if AdminEditedName.text?.trimmingCharacters(in: .whitespacesAndNewlines) != ""{
                database.collection("users").document(id).updateData([
                    "Name": name,
                    ])
                { err in
                    if let err = err {
                        print("Error")
                        return
                    } else {
                        print("The name updated successfully")
                    }
                }
            }
            if AdminEditedPassword.text?.trimmingCharacters(in: .whitespacesAndNewlines) != ""{
                database.collection("users").document(id).updateData([
                    "Pass": password,
                    ])
                { err in
                    if let err = err {
                        print("Error")
                        return
                    } else {
                        print("The password updated successfully")
                    }
                }
            }
            self.transitionToViewAdimnProfile()
        }
    }
```

For Editing User Info:

```
class EditUserProfileViewController: UIViewController {
    @IBOutlet weak var EditUserError: UILabel!
    @IBOutlet weak var UserEditedName: UITextField!
    @IBOutlet weak var UserEditedPassword: UITextField!
    @IBOutlet weak var UserEditedDueDate: UITextField!
    @IBOutlet weak var UpdateUserInfo: UIButton!
    @IBOutlet weak var BackButton: UIButton!
    override func viewDidLoad() {
        super.viewDidLoad()
        setUpElements()
    }
    func setUpElements (){
        EditUserError.alpha = 0
    }
    @IBAction func EditUserInfo(_ sender: Any) {
        let error = validateFields()
        if error != nil {
            showError(error!)
        }else {
            let db = Firestore.firestore()
            let user = Auth.auth().currentUser
            let id = user.uid
            let name = UserEditedName.text!.trimmingCharacters(in: .whitespacesAndNewlines)
            let password = UserEditedPassword.text!.trimmingCharacters(in:
                .whitespacesAndNewlines)
            let duedate = UserEditedDueDate.text!.trimmingCharacters(in:
                .whitespacesAndNewlines)
            let database = Firestore.firestore()
            if UserEditedName.text?.trimmingCharacters(in: .whitespacesAndNewlines) != ""{
                database.collection("users").document(id).updateData([
                    "Name": name,
                    ])
                { err in
                    if let err = err {
                        print("Error")
                        return
                    } else {
                        print("The name updated successfully")
                    }
                }
             }
          if UserEditedPassword.text?.trimmingCharacters(in: .whitespacesAndNewlines) != ""{
              database.collection("users").document(id).updateData([
                  "Pass": password,
                  ])
              { err in
                  if let err = err {
                      print("Error")
                      return
                  } else {
                      print("The password updated successfully")
                  }
              }
          }
          if UserEditedDueDate.text?.trimmingCharacters(in: .whitespacesAndNewlines) != ""{
              database.collection("users").document(id).updateData([
                  "Date": duedate,
                  ])
              { err in
                  if let err = err {
                      print("Error")
                      return
                  } else {
                      print("The date updated successfully")
                  }
              }
          }
          self.transitionToViewUserProfile()
      }
  }
```

For Both :

Validate input and error :

```
func validateFields() -> String? {
    if UserEditedName.text?.trimmingCharacters(in: .whitespacesAndNewlines) == "" &&
        UserEditedPassword.text?.trimmingCharacters(in: .whitespacesAndNewlines) == "" &&
        UserEditedDueDate.text?.trimmingCharacters(in: .whitespacesAndNewlines) == ""  {
        return "You must to edit one field atleast."
    }
    let newName = UserEditedName.text!.trimmingCharacters(in: .whitespacesAndNewlines)
    if newName.count > 0 {
    if Utilities.isName(newName) == false {
        return "Please make sure you enter a valid name."
    }
    }
    let newDuedate = UserEditedDueDate.text!.trimmingCharacters(in:
        .whitespacesAndNewlines)
    if newDuedate.count > 0 {
    if Utilities.isDate(newDuedate) == false {
        return "Please make sure you enter a valid date in (yyyy-mm-dd) format."
    }
    }
    let newPassword = UserEditedPassword.text!.trimmingCharacters(in:
        .whitespacesAndNewlines)
    if newPassword.count > 0 {
    if Utilities.isPasswordValid(newPassword) == false {
        return "Please make sure your password is at least 8 characters, contains a
            special character and a number."
    }
    }
    return nil
}
```

Showing error messages :

```
func showError(_ message:String) {
    EditAdminError.text = message
    EditAdminError.alpha = 1
}
```

Java :

For Editing Admin Info:

```
public class editAdmin extends AppCompatActivity {
    EditText name;
    EditText password;
    String namea;
    String passworda;
    Bundle bundle;
    String email;
    DatabaseHelper helper ;
    long feedbackname;
    long feedbackpass;
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_edit_admin);
        Intent i = getIntent();
        bundle = i.getExtras();
        email = bundle.getString( key: "email");
        helper = new DatabaseHelper( context: this);
        name = findViewById(R.id.nameAdminedit);
        password = findViewById(R.id.pasAedit);
    }
    public void onClickEditAdmin(View v){
        namea = name.getText().toString().trim();
        passworda = password.getText().toString().trim();
        if(namea.length()>0)
            feedbackname=helper.updateUserName(email,namea);
        if(passworda.length()>0)
            feedbackpass=helper.updateUserPass(email,passworda);
        if(feedbackname!=0 || feedbackpass!=0 ){
            Toast.makeText( context: this,  text: "Your info has been updated successfully.",
                    Toast.LENGTH_LONG).show();
            Intent intent = new Intent( packageContext: this, AdminAccount.class);
            intent.putExtra( name: "email", email);
            startActivity(intent);
        }
        else {
            Toast.makeText( context: this,  text: "Sorry! something went wrong", Toast.LENGTH_LONG).show();
        }
    }
}
```

For Editing User Info:

```
public class editUser extends AppCompatActivity {
    EditText name;
    EditText password;
    EditText date;
    String namea;
    String passworda;
    String datea;
    Bundle bundle;
    String email, email2;
    DatabaseHelper helper ;
    long feedbackname;
    long feedbackpass;
    long feedbackdate;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_edit_user);
        Intent i = getIntent();
        bundle = i.getExtras();
        email = bundle.getString( key: "email");
        helper = new DatabaseHelper( context: this);
        name = findViewById(R.id.nameedit);
        password = findViewById(R.id.pasedit);
        date = findViewById(R.id.duedateedit);
    }
    public void onClickEditUser(View v){
        namea = name.getText().toString().trim();
        passworda = password.getText().toString().trim();
        datea = date.getText().toString().trim();

        if(namea.length()>0)
            feedbackname=helper.updateUserName(email,namea);
        if(passworda.length()>0)
            feedbackpass=helper.updateUserPass(email,passworda);
        if(datea.length()>0){
            if(!isValid(date.getText().toString())){
                Message( title: "Error",  message: "Make sure the date must be in (yyyy-mm-dd) format");
                date.requestFocus();
                return;
            }
            feedbackdate=helper.updateUserDate(email,datea);
        }
        if(feedbackname!=0 || feedbackpass!=0  || feedbackdate!=0){
            Toast.makeText( context: this,  text: "Your info has been updated successfully.",
                    Toast.LENGTH_LONG).show();
            Intent intent = new Intent( packageContext: this, account.class);
            intent.putExtra( name: "email", email);
            startActivity(intent);
        }
        else {
            Toast.makeText( context: this,  text: "Sorry! something went wrong",
                    Toast.LENGTH_LONG).show();

        }

    }
```

Helper for Both :

```
public long updateUserName(String Email, String name)
{
    ContentValues cv = new ContentValues();
    cv.put("name",name);
    SQLiteDatabase db = this.getWritableDatabase();
    return db.update( table: "user",cv, whereClause: "emaill="+ "'" + Email + "'" ,   whereArgs: null );
}
public long updateUserDate(String Email, String date)
{
    ContentValues cv = new ContentValues();
    cv.put("date",date);
    SQLiteDatabase db = this.getWritableDatabase();
    return db.update( table: "user",cv, whereClause: "emaill="+ "'" + Email + "'" ,   whereArgs: null );
}
public long updateUserPass(String Email, String pass)
{
    ContentValues cv = new ContentValues();
    cv.put("pass",pass);
    SQLiteDatabase db = this.getWritableDatabase();
    return db.update( table: "user",cv, whereClause: "emaill="+ "'" + Email + "'" ,   whereArgs: null );
}
```

Date format :

```
// source https://stackoverflow.com/questions/2149680/regex-date-format-validation-on-java
public static boolean isValid(String text) {
    if (text == null || !text.matches( regex: "\\d{4}-[01]\\d-[0-3]\\d"))
        return false;
    SimpleDateFormat df = new SimpleDateFormat( pattern: "yyyy-MM-dd");
    df.setLenient(false);
    try {
        df.parse(text);
        return true;
    } catch (ParseException ex) {
        return false;
    }
}
}
```

Showing message :

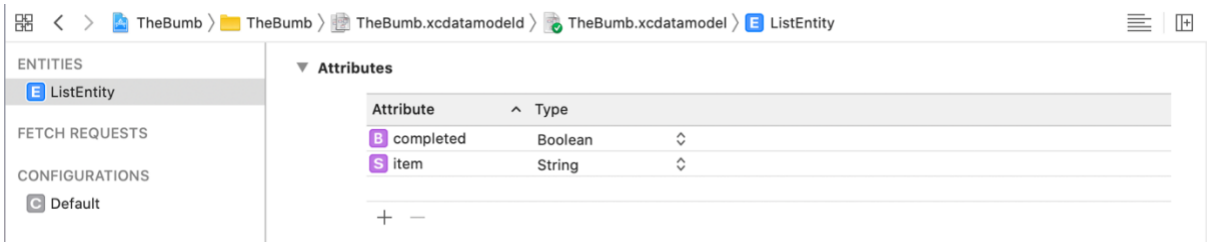
```
public void Message(String title, String message)
{
    AlertDialog.Builder builder=new AlertDialog.Builder( context: this);
    builder.setCancelable(true);
    builder.setTitle(title);
    builder.setMessage(message);
    builder.show();
}
```

Appointment Page:

In swift the appointments were saved in Core Data file which is named in our project as ListEntity that has two attributes the first one is item which saves the appointment's name and the second one is completed which is a Boolean value weather the appointment checked as completed or not, while in java the appointments were saved SQL database.

Also, the dialog in swift was created using UIAlertController class, however in java the dialog created by using Alert Dialog class. Further, in swift the appointment was deleted by using remove function that supported by core data, while in java the appointment deleted by using delete function that supported by SQLite Open Helper class.

Swift:




```
123         func saveItem(itemToSave : String){
124             _ = UIApplication.shared.delegate as! AppDelegate
125             let managedContext = (UIApplication.shared.delegate as! AppDelegate).persistentContainer.viewContext
126             let entity = NSEntityDescription.entity(forEntityName: "ListEntity", in: managedContext)
127             let item = NSManagedObject(entity: entity! , insertInto: managedContext)
128             item.setValue(itemToSave, forKey: "item")
129
130
131             do{
132                 try managedContext.save()
133                 listItems.append(item)
134             }catch{
135                 print("Erorr")
136             }
137         }
```

```swift
@IBAction func AddButtonPressed(_ sender: UIButton) {

    var textField = UITextField()
                let alertController = UIAlertController(title: "Add new appointment", message: "",
                    preferredStyle: .alert)
                let confirmAction = UIAlertAction(title: "Add", style: .default, handler: ({
                    (_) in
                    if let field = alertController.textFields![0] as? UITextField  {
                        self.saveItem(itemToSave: field.text!)
                        self.tableView.reloadData()
                    }
                    }
                ))

                let cancelAction = UIAlertAction(title: "Cancel", style: .cancel, handler: nil)
                alertController.addTextField{ (field) in
                    textField = field
                    textField.placeholder = " add new appointments"
                }
                alertController.addAction(confirmAction)
                alertController.addAction(cancelAction)

                self.present(alertController, animated: true, completion: nil)
}
```

```swift
override func tableView(_ tableView: UITableView, commit editingStyle: UITableViewCell.EditingStyle,
    forRowAt indexPath: IndexPath) {

    if editingStyle==UITableViewCell.EditingStyle.delete{
        context.delete(self.listItems[indexPath.row])
        do{
            try context.save()
            self.listItems.remove(at: indexPath.row)
            tableView.deleteRows(at: [indexPath], with: UITableView.RowAnimation.automatic)
             self.tableView.reloadData()
        }catch{
            print("Error")
        }

    }
}
```

Java:

```java
@Override
public  boolean onOptionsItemSelected(MenuItem item){

    switch (item.getItemId()){

        case R.id.action_add_task:
            final EditText taskEditText= new EditText( context: this);

            AlertDialog alertDialog= new AlertDialog.Builder( context: this)
                    .setTitle("Add new appointment")
                    .setMessage(" ")
                    .setView(taskEditText)

                    .setPositiveButton( text: "Add", new DialogInterface.OnClickListener() {
                        @Override
                        public void onClick(DialogInterface dialog, int which) {
                            String task=String.valueOf(taskEditText.getText());
                            SQLiteDatabase db= taskHelper.getWritableDatabase();

                            ContentValues values= new ContentValues();
                            values.put(Task.taskEntry.COL_TASK_TITLE,task);
                            db.insertWithOnConflict(Task.taskEntry.TABLE, nullColumnHack: null,values,SQLiteDatabase.CONFLICT_REPLACE);
                            db.close();
                            updateUI();
                        }
                    })
                    .setNegativeButton( text: "Cancel", listener: null)
                    .create();
            alertDialog.show();
            return true;

        default:
            return super.onOptionsItemSelected(item);
    }
}
```

```
150
151   @   public void deleteAppointment(View view){
152
153           View parent = (View) view.getParent();
154           TextView taskTextView = (TextView) parent.findViewById(R.id.task_title);
155           String task = String.valueOf(taskTextView.getText());
156           SQLiteDatabase db = taskHelper.getWritableDatabase();
157           db.delete(Task.taskEntry.TABLE,  whereClause: Task.taskEntry.COL_TASK_TITLE + "= ?",new String[]{task});
158           db.close();
159           updateUI();
160       }
```

Reference for the source code:

https://drive.google.com/drive/folders/1s6QowON9GRhm6hEUBv3z1gpTypViMU1R?usp=sharing