

Control de documento

Nombre del proyecto	Healthec
Cierre de iteración	I4 17/03/2023
Generador por	Carlos Antonio Madrigal Trejo
Aprobado por	Carlos Antonio Madrigal Trejo
Alcance de la distribución del documento	Control interno para todo el proyecto.

Índice

Sobre este documento	3
Resumen de la Iteración	4
Identificación	4
Hitos especiales	4
Artefactos y evaluación	4
Riesgos y problemas (Riesgo con su identificador si sucedió o no sucedió)	5
Asignación de recursos	5
Anexos	6
Anexo A.	6
PROG-01 - Programar la base de datos	6
Anexo B.	9
Anexo C.	12
Glosario de términos	15

Sobre este documento

La calidad se logra por medio de la revisión constante de las actividades que conducen desde la idea al producto. Al momento del cierre de una iteración es buen momento para hacer un alto, y evaluar lo logrado, los problemas encontrados y los retos a enfrentar.

El presente documento marca el final de la iteración I4, y contiene una evaluación de los artefactos y actividades realizadas durante la misma.

Se recogen también las impresiones y observaciones hechas durante el desarrollo de la iteración, así como el esfuerzo invertido en cada una de las disciplinas involucradas.

Resumen de la Iteración

Identificación

Código de la iteración	Fase a la que pertenece	Fecha de inicio	Fecha de cierre	Comentarios
I4	Implementación	13/03/23	17/03/23	Se completó correctamente.

Hitos especiales

En este sprint se logró crear los diseños de la base de datos, página web e interfaces requeridos para la posterior creación de la aplicación en código.

Artefactos y evaluación

Artefacto	Meta (%)	Comentarios
PROG-01	100%	Se logró programar la base de datos
PROG-02	80%	Se lograron los puntos imprescindibles de esta tarea, sin embargo se puede desarrollar más en el siguiente sprint.

Artefacto	Aspecto a evaluar	Evaluación	Comentarios
PROG-01	Programación	100%	Se cumplió en su totalidad y en tiempo la tarea.
PROG-02	Programación	80%	Se logró avanzar considerablemente, se retomará en el siguiente sprint las tareas mínimas faltantes.

Riesgos y problemas (Riesgo con su identificador si sucedió o no sucedió)

Notas y observaciones

	Riesgo	Ocurrió
RIE-02	Fallas de hardware	No
RIE-04	Enfermedades	No
RIE-10	Falta de ética y moral del personal	No
RIE-12	Requisitos confusos o ambiguos	No
RIE-14	Ambiente laboral deficiente	No
RIE-16	Documentación deficiente	No
RIE-19	Estimación del tiempo inadecuada	No
RIE-21	Falta de comunicación con el cliente	No
RIE-22	Falta de claridad en los roles de actividades	No
RIE-23	Falta de experiencia del líder de proyecto	No

Asignación de recursos

Rol	Horas-Hombre	Desempeñado por(área)	Observaciones
Programador	4 horas.	Carlos Antonio Madrigal Trejo	Creó y desarrolló una muy buena base de datos
Programador	2 horas.	Fernando Pérez Romero	Investigó acerca de la sintaxis de la base de datos.
Programador	2 horas.	Omar Adrián Tapia Guzmán	Realizó la interfaz del login correctamente.
Programador	2 horas.	Rubén Dario Vidaña	Realizó un buen desempeño en la modificación de la base de datos.
Programador	2 horas.	Carlos Daniel López Romo	Apoyo en el desarrollo de la base de datos.

Anexos

Anexo A.

PROG-01 - Programar la base de datos

DataBaseHealthec.java

```
package mx.GPS.healthec;

import android.content.ContentValues;
import android.content.Context;
import android.database.Cursor;
import android.database.sqlite.SQLiteDatabase;
import android.database.sqlite.SQLiteOpenHelper;

import androidx.annotation.Nullable;

import java.util.ArrayList;
import java.util.List;

public class DataBaseHealthec extends SQLiteOpenHelper {

    //En esta seccion se definen las constantes que se utilizaran para referirnos a la tabla
    y las columnas de la base de datos
    //-----
    // En esta parte definimos la constantes a utilizar en la tabla usuario y sus columnas
    public static final String TABLA_USUARIO = "TABLA_USUARIO";
    public static final String COLUMNA_USUARIO_CORREO = "USUARIO_CORREO";
    public static final String COLUMNA_USUARIO_CLAVE = "USUARIO_CLAVE";
    public static final String COLUMNA_USUARIO_ID = "USUARIO_ID";
    //-----
    //En esta parte definimos la constantes a utilizar en la tabla recordatorios y sus
    columnas
    public static final String TABLA_RECORDATORIOS = "TABLA_RECORDATORIOS";
    public static final String COLUMNA_RECORDATORIOS_CLINICA =
"COLUMNA_RECORDATORIOS_CLINICA";
    public static final String COLUMNA_RECORDATORIOS_MEDICO =
"COLUMNA_RECORDATORIOS_MEDICOS";
    public static final String COLUMNA_RECORDATORIOS_FECHAHORA =
"COLUMNA_RECORDATORIOS_FECHAHORA";
    public static final String COLUMNA_RECORDATORIOS_ID = "COLUMNA_RECORDATORIOS_ID";
    public static final String COLUMNA_HORARIOSUEÑO_ID = "COLUMNA_HORARIOSUEÑO_ID";
    //-----
    //En esta parte definimos la constantes a utilizar en la tabla horario, sueño y sus
    columnas
    public static final String TABLA_HORARIOSUEÑO = "TABLA_HORARIOSUEÑO ";
    public static final String COLUMNA_HORARIOSUEÑO_DIA = "COLUMNA_HORARIOSUEÑO_DIA";
    public static final String COLUMNA_HORARIOSUEÑO_HORA = "COLUMNA_HORARIOSUEÑO_HORA";
    //-----
    //En esta parte definimos la constantes a utilizar en la tabla recetarios y sus
    columnas
    public static final String TABLA_RECETARIOS = "TABLA_RECETARIOS";
    public static final String COLUMNA_RECETARIOS_ID = "COLUMNA_RECETARIOS_ID";
    public static final String COLUMNA_RECETARIOS_RECETA = "COLUMNA_RECETARIOS_RECETA";
    public static final String COLUMNA_RECETARIOS_PASOS = "COLUMNA_RECETARIOS_PASOS";
    //-----
    //En esta parte definimos la constantes a utilizar en la tabla consejos y sus columnas
    public static final String TABLA_CONSEJOS = "TABLA_CONSEJOS";
    public static final String COLUMNA_CONSEJOS_ID = "COLUMNA_CONSEJOS_ID";
    public static final String COLUMNA_CONSEJOS_DESCRIPCION =
"COLUMNA_CONSEJOS_DESCRIPCION";

    public DataBaseHealthec(@Nullable Context context) {
        super(context, "healthec.db", null, 1);
    }

    //Esto es llamado la primera vez que la base de datos es accedida. Aqui va el codigo
    para crear
    //una nueva db
    @Override
```

```

public void onCreate(SQLiteDatabase db) {
    //Aquí se crean las tablas de la base de datos y se definen los tipos de datos de
    las columnas
    String createTableUsuario = "CREATE TABLE " + TABLA_USUARIO + " (" +
    COLUMNA_USUARIO_ID + " INTEGER PRIMARY KEY AUTOINCREMENT, " + COLUMNA_USUARIO_CORREO + "
    TEXT, " + COLUMNA_USUARIO_CLAVE + " TEXT)";
    String createTableRecordatorios = "CREATE TABLE " + TABLA_RECORDATORIOS + " (" +
    COLUMNA_RECORDATORIOS_ID + " INTEGER PRIMARY KEY AUTOINCREMENT, " +
    COLUMNA_RECORDATORIOS_MEDICO + " TEXT, " + COLUMNA_RECORDATORIOS_FECHAHORA + "
    TEXT, "+COLUMNA_RECORDATORIOS_CLINICA+"TEXT)";
    String createTableRecetarios = "CREATE TABLE " + TABLA_RECETARIOS + " (" +
    COLUMNA_RECETARIOS_ID + " INTEGER PRIMARY KEY AUTOINCREMENT, " + COLUMNA_RECETARIOS_RECETA
    + " TEXT, " + COLUMNA_RECETARIOS_PASOS + " TEXT)";
    String createTableHorarioSueño = "CREATE TABLE " + TABLA_HORARIOSUEÑO + " (" +
    COLUMNA_HORARIOSUEÑO_ID + " INTEGER PRIMARY KEY AUTOINCREMENT, " + COLUMNA_HORARIOSUEÑO_DIA
    + " TEXT, " + COLUMNA_HORARIOSUEÑO_HORA + " TEXT)";
    String createTableConsejos = "CREATE TABLE " + TABLA_CONSEJOS + " (" +
    COLUMNA_CONSEJOS_ID + " INTEGER PRIMARY KEY AUTOINCREMENT, " + COLUMNA_CONSEJOS_DESCRIPCION
    + " TEXT)";
    db.execSQL(createTableUsuario);
    db.execSQL(createTableConsejos);
    db.execSQL(createTableRecetarios);
    db.execSQL(createTableRecordatorios);
    db.execSQL(createTableHorarioSueño);
}
//Este metodo es llamado si la version de la base de datos cambia. Previene que los
usuarios que
//tengan una version anterior de la bd creasheen cuando se hacen cambios a la db
@Override
public void onUpgrade(SQLiteDatabase db, int i, int i1) {

}

public boolean addOne (UserModel user ){
    SQLiteDatabase db = this.getWritableDatabase();

    ContentValues cv = new ContentValues();

    cv.put(COLUMNA_USUARIO_CORREO, user.getEmail());
    cv.put(COLUMNA_USUARIO_CLAVE, user.getPassword());

    long insert = db.insert(TABLA_USUARIO, null , cv );
    if( insert == -1){
        return false;
    } else {
        return true;
    }
}

//Metodo para obtener todos los registros de una tabla en una Lista.
public List<UserModel> getEveryone() {
    List<UserModel> returnList = new ArrayList<>();
    //Toma la información de la db
    String queryString = "SELECT * FROM " + TABLA_USUARIO;

    SQLiteDatabase db = this.getReadableDatabase();

    Cursor cursor = db.rawQuery(queryString, null); //el cursor es el set de resultados

    if(cursor.moveToFirst()){
        //Se hace un bucle a traves del cursor y crea un nuevo objeto de UserModel los
        cuales se pondran en la lista que se retorna.
        do{
            int userID = cursor.getInt(0);
            String userEmail = cursor.getString(1);
            String userPassowrd = cursor.getString(2);

            UserModel user = new UserModel(userID, userEmail, userPassowrd);
            returnList.add(user);

        } while (cursor.moveToNext());
    } else {
        //Fallo, no anade nada a la lista
    }
}

```

```
        cursor.close();
        db.close();
        return returnList;
    }

    public boolean Exists( UserModel user ){
        String queryString = "SELECT " + COLUMNA_USUARIO_ID + " FROM " + TABLA_USUARIO +
"WHERE " + COLUMNA_USUARIO_CORREO + " = " + user.getEmail() +
        " AND " + COLUMNA_USUARIO_CLAVE + " = " + user.getPassword();

        SQLiteDatabase db = this.getReadableDatabase();

        Cursor cursor = db.rawQuery(queryString, null);
        boolean exist;

        if( cursor.moveToFirst() ){
            exist = true;
        } else {
            exist = false;
        }

        cursor.close();
        db.close();

        return exist;
    }
}
```


Anexo B.

PROG-02 Programar el Login Activity

LoginActivity.java

```
package mx.GPS.healthec;
import androidx.appcompat.app.AppCompatActivity;

import android.content.Intent;
import android.os.Bundle;
import android.view.View;
import android.widget.Button;
import android.widget.EditText;
import android.widget.Toast;

public class LoginActivity extends AppCompatActivity {

    //Referencia a los botones y otros controles en el layout
    Button btn_registrar, btn_ingresar, btn_recuperar;
    EditText et_email, et_password;

    //METODO ONCREATE, SE EJECUTA CUANDO LOGINACTIVITY SE INICIA
    @Override
    //Llama al método onCreate() de la clase base Activity utilizando el parámetro
    savedInstanceState.
    //este método establece el diseño de la actividad y busca los elementos de la interfaz
    de
    // usuario para poder interactuar con ellos posteriormente en la aplicación.

    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_login);
        //Se le asigna a las variables el elemento creado por la computadora en el
    layout-----//
        btn_ingresar = findViewById(R.id.btn_ingresar);
        btn_registrar = findViewById(R.id.btn_registrar);
        btn_recuperar = findViewById(R.id.btn_recuperar);

        et_email = findViewById(R.id.et_email);
        et_password = findViewById(R.id.et_password);

        //Listeners de los
    botones-----//
        btn_ingresar.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View view) {
                //SE DECLARA UNA INSTANCIA DE UserModel llamada usuario
                UserModel usuario;
                //Se intenta crear una nueva instancia de UserModel utilizando los valores
    ingresados
                // en los campos de correo electrónico y contraseña de la interfaz de
    usuario.
                // Si se produce una excepción, se muestra un mensaje de error y se crea
    una instancia de
                // UserModel con valores predeterminados ("error" y "-1").
                try{
                    usuario = new UserModel(-1, et_email.getText().toString(),
                        et_password.getText().toString());
                    Toast.makeText(LoginActivity.this, usuario.toString(),
    Toast.LENGTH_SHORT).show();
                } catch (Exception e){
                    Toast.makeText(LoginActivity.this, "Es necesario rellenar todos los
    campos",
                        Toast.LENGTH_SHORT).show();
                    usuario = new UserModel(-1, "error", "error");
                }
            }
        });
    }
```



```
//Se declara una instancia de DataBaseHealthec, que es una clase que maneja
la base de datos
// SQL utilizada por la aplicación.
DataBaseHealthec dataBaseHealthec = new
DataBaseHealthec(LoginActivity.this);

//Se llama al método "addOne" de la instancia de DataBaseHealthec, pasando
como argumento
// la instancia de UserModel creada anteriormente. Este método intenta
agregar el usuario a
// la base de datos y devuelve un valor booleano que indica si la operación
fue exitosa o no.
boolean exist = dataBaseHealthec.addOne(usuario);

if( exist ){
//codigo para entrar al menu principal donde se encuentran todas las
opciones
startActivity( new Intent(LoginActivity.this, MainActivity.class));
} else {
//codigo para representar que el usuario no existe
Toast.makeText(LoginActivity.this, "No existe ninguna cuenta con esos
datos",
Toast.LENGTH_LONG).show();
}
});

//-----/
/
btn_registrar.setOnClickListener(new View.OnClickListener() {

//METODO ONCLICK este método se encarga de crear un nuevo usuario con los
valores
// ingresados en la interfaz de usuario y agregarlo a la base de datos de la
aplicación.
@Override
public void onClick(View view) {

UserModel usuario;
try{
usuario = new UserModel(-1, et_email.getText().toString(),
et_password.getText().toString());
Toast.makeText(LoginActivity.this, usuario.toString(),
Toast.LENGTH_SHORT).show();
} catch( Exception e){
Toast.makeText(LoginActivity.this, "Es necesario rellenar todos los
campos",
Toast.LENGTH_SHORT).show();
usuario = new UserModel(-1, "error", "error");
}
DataBaseHealthec dataBaseHealthec = new
DataBaseHealthec(LoginActivity.this);

boolean success = dataBaseHealthec.addOne(usuario);

Toast.makeText(LoginActivity.this, "Success= " + success,
Toast.LENGTH_SHORT).show();

}
});

//-----/
/
btn_recuperar.setOnClickListener(new View.OnClickListener() {
@Override
public void onClick(View view) {

}
});
}
```

UserModel.java

```
package mx.GPS.healthec;

public class UserModel {

    //Atributos de la clase usuario
    private int idUser;
    private String email;
    private String password;
    //Constructor de la clase Usuario
    public UserModel(int idUser, String email, String password) {
        this.idUser = idUser;
        this.email = email;
        this.password = password;
    }
    //Constructor vacío de la clase usuario
    public UserModel() {
    }

    //toString es necesario para imprimir el contenido de objetos de la clase
    @Override
    public String toString() {
        return "UserModel{" +
            "idUser=" + idUser +
            ", email='" + email + '\'' +
            ", password='" + password + '\'' +
            '}';
    }

    //getters and setters de los atributos de la clase
    public int getIdUser() {
        return idUser;
    }

    public void setIdUser(int idUser) {
        this.idUser = idUser;
    }

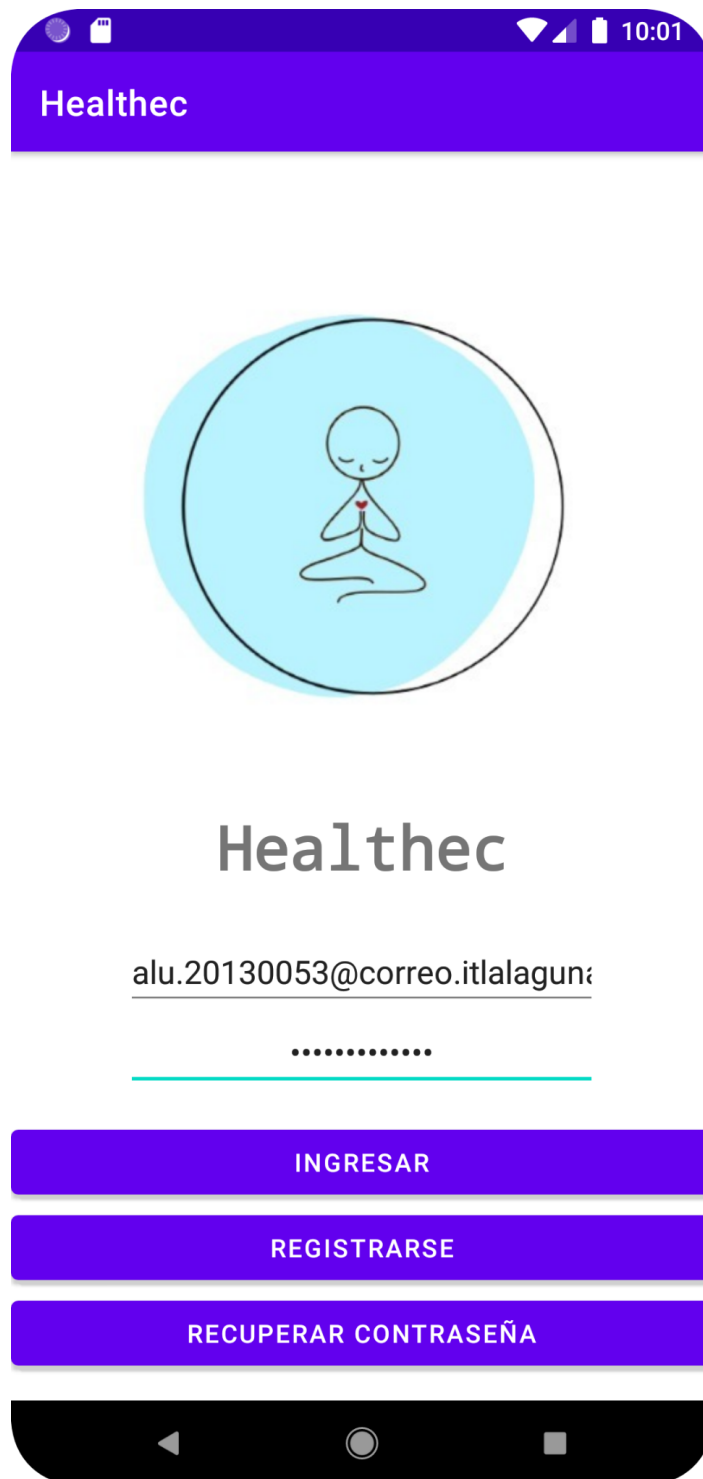
    public String getEmail() {
        return email;
    }

    public void setEmail(String email) {
        this.email = email;
    }

    public String getPassword() {
        return password;
    }

    public void setPassword(String password) {
        this.password = password;
    }

}
```



The image shows a mobile application interface for 'Healthec'. At the top is a purple header bar with the app name 'Healthec' in white. Below the header is a large circular logo featuring a light blue background with a white outline of a person in a meditative pose. Underneath the logo, the word 'Healthec' is written in a large, dark grey font. Below the app name is a text input field containing the email address 'alu.20130053@correo.itlalaguna.mx', followed by a password field represented by a series of dots. A horizontal teal line separates the input fields from the login options. There are three purple buttons stacked vertically: 'INGRESAR' (Login), 'REGISTRARSE' (Register), and 'RECUPERAR CONTRASEÑA' (Recover Password). At the bottom of the screen is a black navigation bar with three icons: a back arrow, a home circle, and a recent apps square.

Healthec

alu.20130053@correo.itlalaguna.mx

.....

INGRESAR

REGISTRARSE

RECUPERAR CONTRASEÑA

Anexo C.

Tabla de McCall

CAPACIDAD	FACTOR	Métrica	Calificación
Operación	Corrección: Grado de cumplimiento de las especificaciones y objetivos del usuario	Compleción: Grado en que se logró implementar en la app los requerimientos especificados por las necesidades del usuario final	5
		Consistencia: Los requerimientos del usuario final cumplen con las técnicas de documentación por norma	3
		Trazabilidad: Los elementos del diseño de la app se identifican a partir de los requerimientos del usuario final	4
	Confiabilidad: Grado en el sistema está disponible para usarse.	Complejidad: La complejidad de las funciones de la app interfieren con la disponibilidad para usarse	4
		Consistencia: El diseño de la app permite el usuario utilizarla en cualquier momento	5
		Tolerancia a errores: Las fallas en la disponibilidad de la app interfieren en las necesidades de los usuarios	3
	Usabilidad: Grado de esfuerzo necesario que se requiere para aprender a utilizarlo.	Facilidad de formación: Facilidad en que el usuario puede aprender a utilizar la app	5
		Operatividad: La app cuenta con una guía para su facilidad de operación	3
	Integridad o Seguridad: Grado en el que se controla el acceso al programa o los datos por usuarios no autorizados.	Facilidad de auditoría: La app cuenta con las autenticaciones necesarias para impedir a usuarios no autorizados acceder a información sensible	4
		Instrumentación: La app cuenta con herramientas para identificar automáticamente brechas de seguridad en la información del usuario	5
		Seguridad: La app cuenta con elementos de protección para asegurar la protección de información importante.	5
	Eficiencia o Performance: Cantidad de recursos y código requeridos por un programa para realizar una función.	Concisión: Nivel de optimización de código en las funciones de la app.	3
		Eficiencia de ejecución: Nivel de eficiencia de ejecución en las funciones de la app.	4
		Operatividad: Nivel de facilidad de operación en las	4

		funciones de la app.	
Transición	Portabilidad: Grado que mide el esfuerzo para migrar un programa de un entorno de operación a otro.	Auto documentación: El código de la app cuenta con la claridad necesaria para portar en otro entorno sin documentación.	5
		Generalidad: La app en general es capaz de ser migrada a otro entorno.	3
		Modularidad: El código de la app es capaz de ser migrado por módulos.	3
	Reusabilidad: Grado de esfuerzo requerido para que el programa o una de sus partes pueda ser utilizado en otro proyecto.	Autodocumentación: El código de la app cuenta con la claridad necesaria para que otro desarrollador pueda utilizarlo en otro proyecto.	5
		Independencia hardware: La app o cualquier parte de su código puede ser utilizado en cualquier modelo celular.	4
		Independencia del sistema: La app o cualquier parte de su código puede ser utilizado en cualquier sistema operativo o versión vigente.	3
	Interoperabilidad: Grado de esfuerzo dedicado para que un sistema o programa pueda operar conjuntamente con otro.	Estd. Comunicaciones: Grado de uso de estándares para que la app pueda operar con otro software conjuntamente.	5
		Estandarización de datos: Nivel de manejo de interoperabilidad con otros softwares.	5
Revisión	Facilidad Mantenimiento: Esfuerzo requerido para localizar y corregir un error en un programa en funcionamiento.	Consistencia: Nivel de documentación empleada para reducir el esfuerzo requerido en corrección de errores.	4
		Modularidad: Nivel de modularidad de las funcionalidades para su fácil mantenimiento.	5
		Simplicidad: Nivel de simplicidad del código para su fácil mantenimiento por cualquier desarrollador incluso ajeno al proyecto.	4
	Flexibilidad: Esfuerzo requerido para modificar un software en funcionamiento.	Capacidad de expansión: Grado permitido para ampliar la app en funcionamiento.	3
		Complejidad: Nivel de complejidad para ampliar la app en funcionamiento.	3
		Consistencia: Nivel de documentación empleada para poder ampliar la app en funcionamiento.	5

	Facilidad de Prueba: Grado de esfuerzo requerido para probar un programa verificando que realice adecuadamente sus funciones.	Auto documentación: La app puede ser probada debido a la claridad del código proporcionada por la documentación realizada.	3
		Facilidad de auditoría: Nivel de facilidad de auditoría de las funciones de la app.	4
		Instrumentación: Nivel de aplicación de herramientas que apliquen pruebas automatizadas a la app.	5

Glosario de términos

Hardware: Conjunto de elementos físicos o materiales que constituyen una computadora o un sistema informático.

Software: Conjunto de programas y rutinas que permiten a la computadora realizar determinadas tareas.

Interfaz: En el contexto de la programación, una interfaz es un conjunto de métodos y propiedades que se pueden implementar en una clase para permitir la comunicación con otras clases y componentes. Una interfaz define un contrato que debe cumplirse para que las clases puedan comunicarse entre sí de manera efectiva.

Hash: En el contexto de la seguridad informática, un hash es un valor numérico que se genera a partir de un conjunto de datos utilizando un algoritmo de hash. Los algoritmos de hash son funciones criptográficas que toman una entrada de datos y producen una salida de longitud fija que es única para esa entrada. Los hashes se utilizan para verificar la integridad de los datos, proteger las contraseñas y garantizar la autenticidad de los mensajes.

Salting: El salting (en español, "salteado") es una técnica de seguridad utilizada en la criptografía de contraseñas. Consiste en agregar una cadena aleatoria de datos a la contraseña antes de realizar el hash. El salting hace que las contraseñas sean más difíciles de romper mediante ataques de fuerza bruta o de diccionario, ya que el hash resultante será diferente para cada usuario. El proceso de salting y hashing juntos se conoce como "salting and hashing" o "hashing with salt".

Métodos: En programación, los métodos son bloques de código que realizan una tarea específica. Los métodos pueden recibir parámetros y devolver valores, lo que les permite interactuar con otros componentes del programa. Los métodos se utilizan para organizar y modularizar el código, lo que facilita el mantenimiento y la reutilización del mismo.

Base de datos: Una base de datos es un sistema de almacenamiento y recuperación de información. Las bases de datos se utilizan comúnmente en aplicaciones informáticas para almacenar información estructurada y permitir la recuperación y manipulación de los datos de manera eficiente.

Sensor: Un sensor es un componente de hardware que detecta y responde a cambios en su entorno. En el contexto de la programación móvil, los sensores se utilizan comúnmente en los dispositivos móviles para detectar la ubicación, el movimiento, la luz, la proximidad y otros aspectos del entorno del dispositivo. Los datos del sensor se pueden utilizar para crear experiencias interactivas en tiempo real y para mejorar la precisión de las aplicaciones.

Tareas en segundo plano: Las tareas en segundo plano son operaciones tipo "desencadenar y olvidar" y su progreso de ejecución no tiene ningún impacto en la interfaz de usuario o el proceso de llamada. Esto significa que el proceso de llamada no espera a la finalización de las tareas.