Reporte

14/06/2023

Reporte de actividades Programación de dispositivos Móviles I

Ingeniería en Sistemas Computacionales

Edwin Ernesto Canul Gongora (19390062)

Belen Del Rosario Benito Tecuautzin (19390239)

Grupo I8U

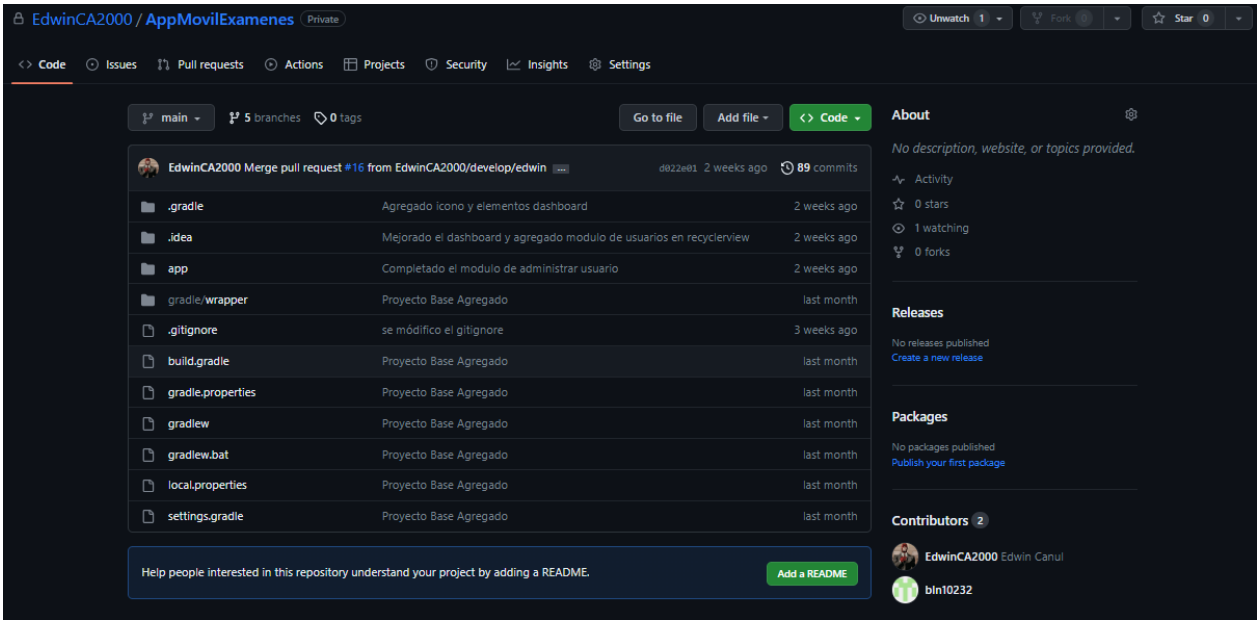
Programación de dispositivos Móviles I

Contenido

Actividades de inicio

Lo primero que se realizo fue crear un repositorio en github para ir desarrollando el proyecto.

URL: [EdwinCA2000/AppMovilExamenes (github.com)]



Para la segunda actividad antes de comenzar a desarrollar código instalamos las librerías necesarias para poder trabajar:

```
dependencies {

    implementation fileTree(dir: "libs", include: ["*.jar"])
    implementation 'androidx.core:core-ktx:1.10.1'
    implementation 'androidx.appcompat:appcompat:1.6.1'
    implementation 'androidx.constraintlayout:constraintlayout:2.1.4'
    implementation 'androidx.legacy:legacy-support-v4:1.0.0'
    implementation 'com.squareup.retrofit2:retrofit:2.9.0'
    implementation 'com.squareup.retrofit2:converter-gson:2.9.0'
    implementation "androidx.preference:preference-ktx:1.2.0"
    implementation "androidx.recyclerview:recyclerview:1.3.0"
    implementation "androidx.cardview:cardview:1.0.0"
    testImplementation 'junit:junit:4.13.2'
    androidTestImplementation 'androidx.test.ext:junit:1.1.5'
    androidTestImplementation 'androidx.test.espresso:espresso-core:3.5.1'

    // Navigation Component
    implementation 'androidx.navigation:navigation-fragment-ktx:2.5.3'
    implementation 'androidx.navigation:navigation-ui-ktx:2.5.3'
}
```

Desarrollo de la aplicación

Para el desarrollo de la aplicación utilizamos como lenguaje principal Kotlin, ya que lo estudiamos mediante los cursos vistos dentro del programa de trabajo para poder hacer las llamadas de las apis relacionadas con el proyecto que estamos desarrollando en la SEQ, se hizo uso de la librería retrofit, y además que la parte de front se hace uso de las librerías constraint layout, recyclerview y cardview para realizar interfaces dinámicas mediante adaptadores. De igual manera se hace uso de los navigations para poder realizar interfaces enlazadas con diferentes acciones.

Backend

Como se menciono anteriormente lo primero que se hizo para ir desarrollando las distintas partes del proyecto se realizo una interfaz en kotlin con la librería retrofit, la estructura de esta interfaz fue primero generar el URL Base, en donde se pondrá liga de nuestro proyecto en funcionamiento.

```
Edwin Canul *
companion object Factory {
    private const val BASE_URL = "http://ec2-54-211-61-57.compute-1.amazonaws.com:8080/Ceneval-1.0-SNAPSHOT/"
    Edwin Canul
    fun create(context: Context): ApiServicio {
        val httpClient = OkHttpClient.Builder() OkHttpClient.Builder
            .addInterceptor { chain ->
                val originalRequest = chain.request()
                val jsessionId = getJsessionIdFromStorage(context)
                Log.d( tag: "ApiServicioJSSESIONID", jsessionId)
                val modifiedRequest = originalRequest.newBuilder()
                    .header( name: "Cookie", jsessionId)
                    .build()
                chain.proceed(modifiedRequest) ^addInterceptor
            } OkHttpClient.Builder
            .build()


        val retrofit = Retrofit.Builder() Retrofit.Builder
            .baseUrl(BASE_URL) Retrofit.Builder
            .client(httpClient)
            .addConverterFactory(GsonConverterFactory.create(gson))
            .build()

        return retrofit.create(ApiServicio::class.java)
```

Un problema con la que lidiamos fue el sistema de sesiones implementado dentro del proyecto que estaba en la web, ya que se almacenaba en las cookies el JSESSIONID,

por lo que tuvimos que generar método para poder obtener y hacer que las llamadas a las Apis funcionen correctamente:

```
Edwin Canul
private fun getJsessionIdFromStorage(context: Context): String {
    val sharedPreferences: SharedPreferences = PreferenceHelper.defaultPrefs(context)
    val cookie = sharedPreferences.getString( key: "JSESSIONID",  defValue: "") ?: ""
    return if (cookie.isNotEmpty()) {
        val cleanedCookie = if (cookie.startsWith( prefix: "JSESSIONID=")) cookie else "JSESSIONID=$cookie"
        cleanedCookie.substringBefore( delimiter: ";")
    } else {
        ""
    }

}
```

Posteriormente despues de haber creado el funcionamiento, debíamos realizar los métodos en donde se llamarían a las apis en la parte de front:

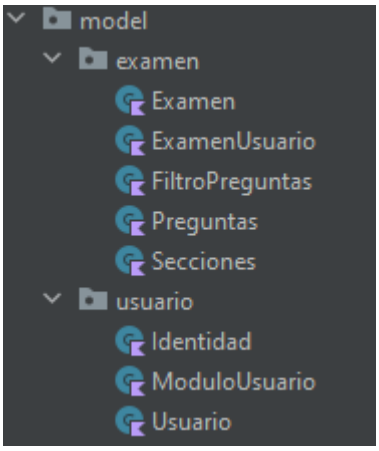
```
interface ApiServicio {

    Edwin Canul
    @GET(value= "tecnologiaeducativa.PerfilExamen/jsonExamenesPerfil")
    fun obtenerExamenUsuario(@Query(value="IdUsuario")idUsuario: Int):
            Call <List<ExamenUsuario>>

    Edwin Canul *
    @POST(value="tecnologieducativa/Sesion/actionLogin")
    fun postLogin(@Query(value="Usuario")usuario: String, @Query(value="Password") password: String):
            Call <LoginRespuesta>

    Edwin Canul
    @GET(value="tecnologiaeducativa.ceneval/jsonTotalPreguntas")
    fun obtenerTotalPreguntas(@Query(value="IdSeccion")idSeccion: Int):
            Call <LoginRespuesta>

    Edwin Canul
    @GET("jsonDatosSesion")
    fun getDatosSesion(): Call<Identidad>
    Edwin Canul
    @GET("tecnologiaeducativa.ceneval/jsonExamenes")
    fun getExamenesDisponibles(): Call<List<Examen>>

    Edwin Canul
    @GET("tecnologiaeducativa.ceneval/jsonSecciones")
    fun getSeccionesExamen(): Call<List<Secciones>>
```

```
@POST(value="tecnologiaeducativa.sesion/actionRegistroUsuario")
fun postRegistro(
    @Query(value="IdUsuario")IdUsuario:Int=1,
    @Query(value="CURP")CURP:String,
    @Query(value="Contrasena")Contrasena:String,
    @Query(value="CorreoElectronico")CorreoElectronico: String,
    @Query(value="Nombres")Nombres: String,
    @Query(value="Apellido1")Apellido1:String,
    @Query(value="Apellido2")Apellido2: String,
):Call<LoginRespuesta>

 Edwin Canul
@POST(value="tecnologiaeducativa.ceneval/actionEditarModuloUsuario")
fun actualizarUsuario(
    @Query(value="IdUsuario")IdUsuario: Int,
    @Query(value="Nombres")Nombres: String,
    @Query(value="Apellido1")Apellido1:String,
    @Query(value="Apellido2")Apellido2: String,
    @Query(value="CURP")CURP:String,
    @Query(value="CorreoElectronico")CorreoElectronico: String,
    @Query(value="Contrasena")Contrasena:String,
    @Query(value="ActivoUsuario")ActivoUsuario:Int,
):Call<ModuloUsuarioRespuesta>
```

```
Edwin Canul
@GET(value="tecnologiaeducativa.ceneval/actionActivarUsuario")
fun actualizarEstadoUser(
    @Query(value="IdUsuario")IdUsuario: Int,
    @Query(value="ActivoUsuario")ActivoUsuario:Int
):Call<RespuestaActivarUser>

Edwin Canul
@GET("tecnologiaeducativa.Sesion/actionCerrarSesion")
fun cerrarSesion(): Call<Unit>

Edwin Canul
@GET("tecnologiaeducativa.ceneval/jsonTotalModuloUsuarios")
fun obtenerTotalUsuariosRegistrados(): Call <LoginRespuesta>

Edwin Canul
@GET("tecnologiaeducativa.ceneval/jsonTotalExamenesUsuarios")
fun obtenerTotalExamenCompletados(): Call <LoginRespuesta>

Edwin Canul
@GET("tecnologiaeducativa.ceneval/jsonExamenesUsuario")
fun obtenerExamenesUsuario(): Call <List<ExamenUsuario>>

Edwin Canul
@GET("tecnologiaeducativa.ceneval/jsonModuloUsuarios")
fun obtenerModuloUsuarios(): Call <List<Usuario>>
```

Y así se hacia el procedimiento para cada vez que queríamos agregar un api de nuestro proyecto web.

Posteriormente para poder almacenar las respuestas en caso que la llamada al api sea exitosa, debíamos tener lista nuestro modelo con los atributos que se necesitaban:

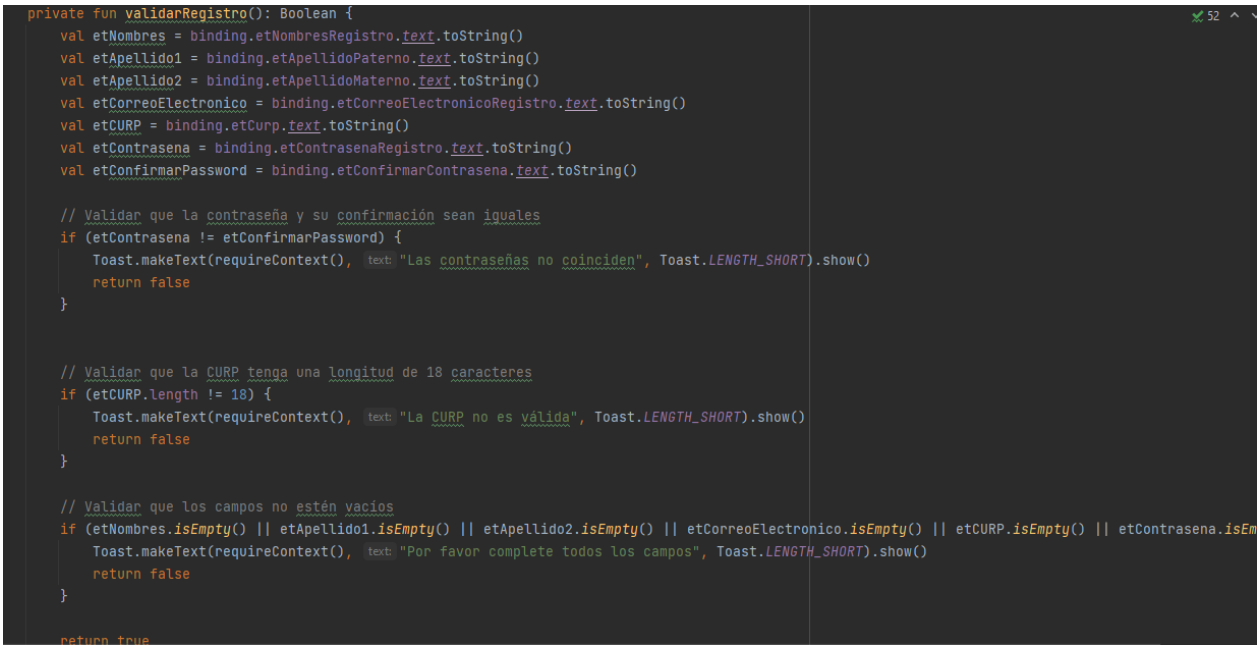


La estructura de estas era de esta manera:

```
package com.example.examenesseq.model.usuario


Edwin Canul
data class Identidad(
    val IdUsuario: Int,
    val CURP: String,
    val CorreoElectronico: String,
    val Nombres: String,
    val Apellido1: String,
    val Apellido2: String,
    val IdPerfil: Int,
    val ActivoUsuario: Int
)
```

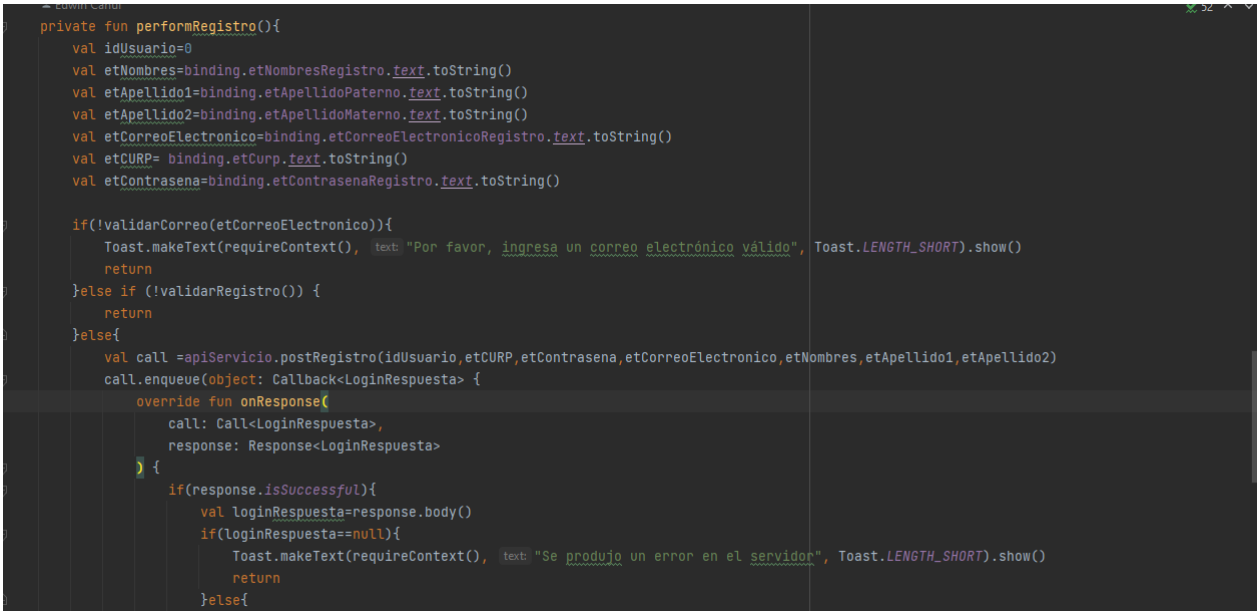
FrontEnd

Una vez realizado el proceso anterior, teníamos nuestros fragmentos listos para poder trabajar con ellos mediante apis, tomemos de ejemplo el fragmento de registro de usuarios: En este fragmento realizamos los métodos necesarios para validar los EditText que hay en el layout:



```
private fun validarRegistro(): Boolean {
    val etNombres = binding.etNombresRegistro.text.toString()
    val etApellido1 = binding.etApellidoPaterno.text.toString()
    val etApellido2 = binding.etApellidoMaterno.text.toString()
    val etCorreoElectronico = binding.etCorreoElectronicoRegistro.text.toString()
    val etCURP = binding.etCurp.text.toString()
    val etContrasena = binding.etContrasenaRegistro.text.toString()
    val etConfirmarPassword = binding.etConfirmarContrasena.text.toString()

    // Validar que la contraseña y su confirmación sean iguales
    if (etContrasena != etConfirmarPassword) {
        Toast.makeText(requireContext(), text: "Las contraseñas no coinciden", Toast.LENGTH_SHORT).show()
        return false
    }


    // Validar que la CURP tenga una longitud de 18 caracteres
    if (etCURP.length != 18) {
        Toast.makeText(requireContext(), text: "La CURP no es válida", Toast.LENGTH_SHORT).show()
        return false
    }

    // Validar que los campos no estén vacíos
    if (etNombres.isEmpty() || etApellido1.isEmpty() || etApellido2.isEmpty() || etCorreoElectronico.isEmpty() || etCURP.isEmpty() || etContrasena.isEm
        Toast.makeText(requireContext(), text: "Por favor complete todos los campos", Toast.LENGTH_SHORT).show()
        return false
    }

    return true
```

Posteriormente creamos el método de performRegistro que es donde se hará uso de una llamada a la api de nuestro proyecto:



```
private fun performRegistro(){
    val idUsuario=0
    val etNombres=binding.etNombresRegistro.text.toString()
    val etApellido1=binding.etApellidoPaterno.text.toString()
    val etApellido2=binding.etApellidoMaterno.text.toString()
    val etCorreoElectronico=binding.etCorreoElectronicoRegistro.text.toString()
    val etCURP= binding.etCurp.text.toString()
    val etContrasena=binding.etContrasenaRegistro.text.toString()

    if(!validarCorreo(etCorreoElectronico)){
        Toast.makeText(requireContext(), text: "Por favor, ingresa un correo electrónico válido", Toast.LENGTH_SHORT).show()
        return
    }else if (!validarRegistro()) {
        return
    }else{
        val call =apiServicio.postRegistro(idUsuario,etCURP,etContrasena,etCorreoElectronico,etNombres,etApellido1,etApellido2)
        call.enqueue(object: Callback<LoginRespuesta> {
            override fun onResponse(
                call: Call<LoginRespuesta>,
                response: Response<LoginRespuesta>
            ) {
                if(response.isSuccessful){
                    val loginRespuesta=response.body()
                    if(loginRespuesta==null){
                        Toast.makeText(requireContext(), text: "Se produjo un error en el servidor", Toast.LENGTH_SHORT).show()
                        return
                    }else{
```

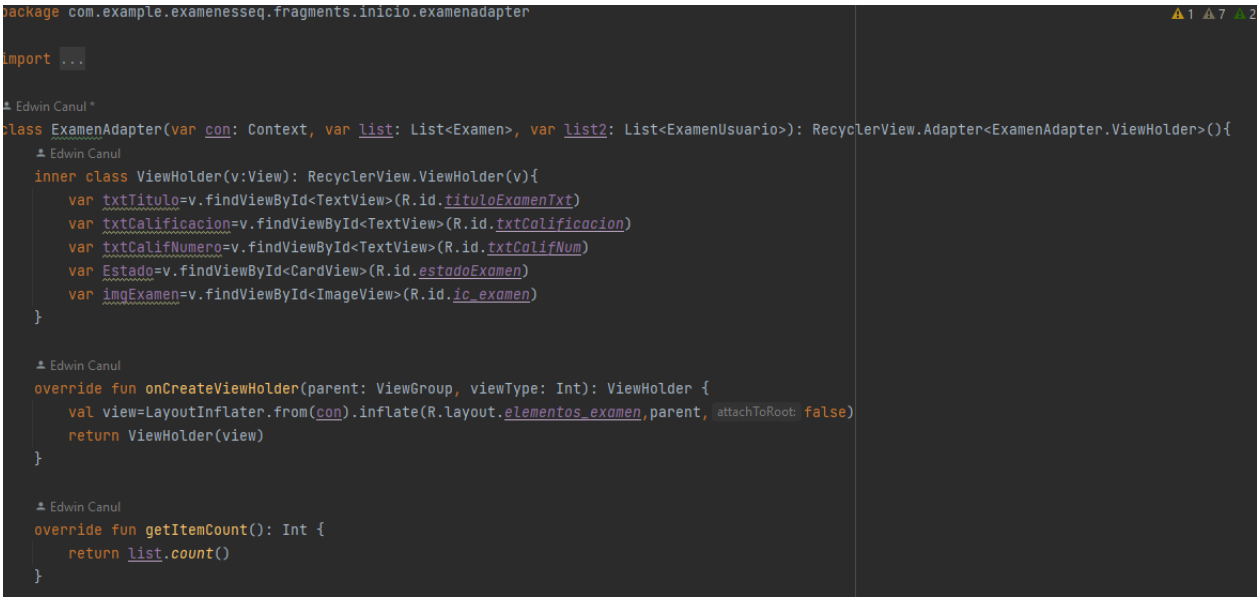
```
                }else{
                    val preferences = PreferenceHelper.defaultPrefs(requireContext())
                    val identidad = loginRespuesta.Objeto
                    val jsessionid = response.headers()["Set-Cookie"] ?: ""
                    Log.d( tag: "JSESSIONID", jsessionid)
                    preferences.setJSessionId(jsessionid)
                    preferences.saveIdentidad(identidad)
                    irAInicio()
                }
            }else{
                Toast.makeText(requireContext(),  text: "Ocurrio un error en el servidor", Toast.LENGTH_SHORT).show()
            }
        }

        override fun onFailure(call: Call<LoginRespuesta>, t: Throwable) {
            Toast.makeText(requireContext(),  text: "Hubo un error en el servidor", Toast.LENGTH_SHORT).show()
        }

    })
  }
}
```

Y de esta manera es con la que se hacia la llamada a las Apis en la parte de frontEnd, como se puede ver en el código se hace uso de sharedPreferences para no tener que realizar las llamadas de las Apis múltiples veces para optimizar nuestro código.

Esta prácticamente fue toda la estructura con la que hacía para cada fragmento.

Adaptadores con recyclerview y cardview

Un caso de ejemplo utilizando los adaptadores fue en el fragmento de inicio que es donde te redirige al momento de registrarte o iniciar sesión, dentro de este fragmento tenemos un recyclerview, en donde desplegamos lo que queremos en un cardview:



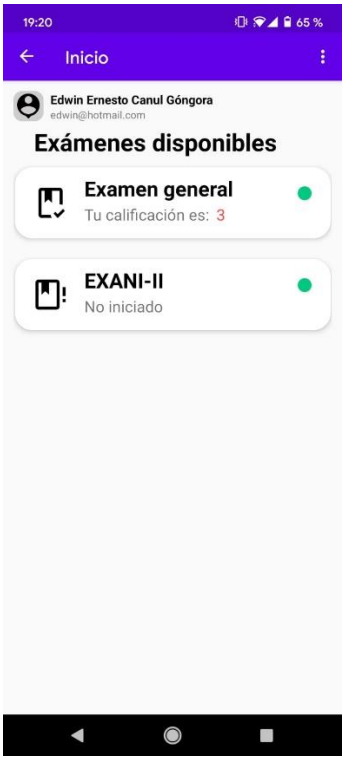
```
package com.example.examenesseq.fragments.inicio.examenadapter

import ...

Edwin Canul *
class ExamenAdapter(var con: Context, var list: List<Examen>, var list2: List<ExamenUsuario>): RecyclerView.Adapter<ExamenAdapter.ViewHolder>(){
    Edwin Canul
    inner class ViewHolder(v:View): RecyclerView.ViewHolder(v){
        var txtTitulo=v.findViewById<TextView>(R.id.tituloExamenTxt)
        var txtCalificacion=v.findViewById<TextView>(R.id.txtCalificacion)
        var txtCalifNumero=v.findViewById<TextView>(R.id.txtCalifNum)
        var Estado=v.findViewById<CardView>(R.id.estadoExamen)
        var imgExamen=v.findViewById<ImageView>(R.id.ic_examen)
    }

    Edwin Canul
    override fun onCreateViewHolder(parent: ViewGroup, viewType: Int): ViewHolder {
        val view=LayoutInflater.from(con).inflate(R.layout.elementos_examen,parent, attachToRoot: false)
        return ViewHolder(view)
    }

    Edwin Canul
    override fun getItemCount(): Int {
        return list.count()
    }
```

```
Edwin Canul
override fun onBindViewHolder(holder: ViewHolder, position: Int) {
    holder.txtTitulo.text = list[position].TituloExamen

    val examen = list[position]
    val examenUsuarioDetails= list2.find { it.IdExamen == examen.IdExamen }

    if (examenUsuarioDetails != null) {
        if(examenUsuarioDetails.Estado==2 || examenUsuarioDetails.Estado==3){
            if (examenUsuarioDetails.TotalCalificacion>60){
                holder.txtCalifNumero.setTextColor(ContextCompat.getColor(con, R.color.green))
                holder.txtCalifNumero.text=examenUsuarioDetails.TotalCalificacion.toString()
            }else{
                holder.txtCalifNumero.text=examenUsuarioDetails.TotalCalificacion.toString()
                holder.txtCalifNumero.setTextColor(ContextCompat.getColor(con, R.color.red))
            }
        }else{
            val avisoEstadoExamen="En proceso"
            holder.txtCalificacion.text=avisoEstadoExamen
            holder.txtCalifNumero.visibility=View.INVISIBLE
        }
    }else{
        val avisoEstadoExamen="No iniciado"
        holder.txtCalificacion.text=avisoEstadoExamen
        holder.txtCalifNumero.visibility=View.INVISIBLE
```

```
}



holder.itemView.setOnClickListener { it: View!
    val detalleExamenModalCompletado = examenUsuarioDetails?.let { it1 -> ExamenModalCompletado(examen, it1) }
    if (detalleExamenModalCompletado != null) {
        detalleExamenModalCompletado.show((con as AppCompatActivity).supportFragmentManager, tag: "DetalleExamenModalCompletado")
    }else{
        val detalleExamenModal = ExamenModal(examen)
        detalleExamenModal.show((con as AppCompatActivity).supportFragmentManager, tag: "DetalleExamenModal")
    }
}
```

```
// Obtener el estado del ExamenUsuario
val estado = examenUsuarioDetails?.Estado

when(estado){
    1 -> holder.imgExamen.setImageResource(R.drawable.book_alert_outline)
    2 -> holder.imgExamen.setImageResource(R.drawable.book_check_outline)
    3 -> holder.imgExamen.setImageResource(R.drawable.book_check_outline)
    else -> holder.imgExamen.setImageResource(R.drawable.book_alert_outline)
}
```

El resultado fue este:



Como se puede ver en la pantalla al darle click a los cardview se abre un modal dependiendo de si el estado del examen es finalizado o no iniciado:

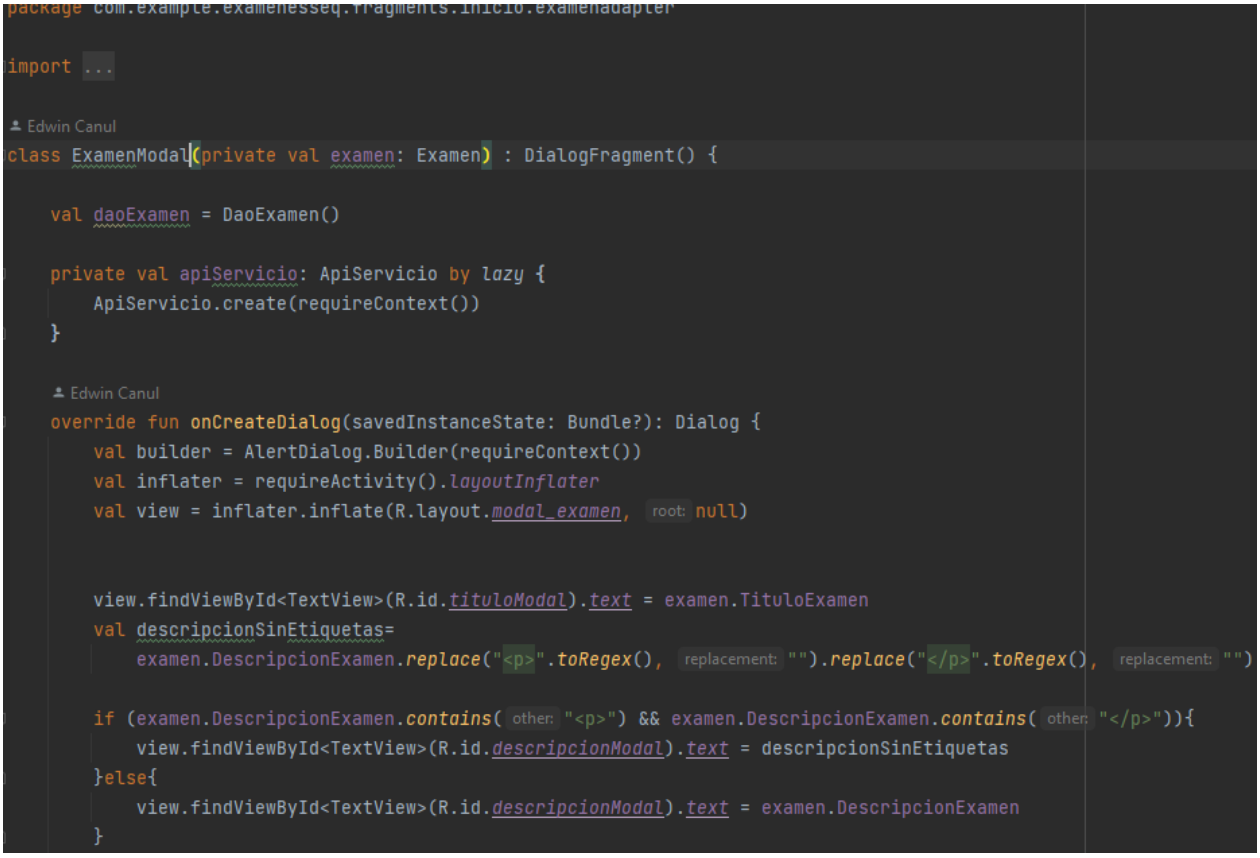
Finalizado:

```
class ExamenModalCompletado(private val examen: Examen, private val examenUsuario: ExamenUsuario) : DialogFragment() {
    private val apiServicio: ApiServicio by lazy {
        ApiServicio.create(requireContext())
    }
    Edwin Canul
    override fun onCreateDialog(savedInstanceState: Bundle?): Dialog {
        val builder = AlertDialog.Builder(requireContext())
        val inflater = requireActivity().layoutInflater
        val view = inflater.inflate(R.layout.modal_examen_completado,  root: null)

        view.findViewById<TextView>(R.id.tituloModal).text = examen.TituloExamen

        val descripcionSinEtiquetas=examen.DescripcionExamen.replace("<p>".toRegex(),  replacement: "")?.replace("</p>".toRegex(),  replacement: "")

        if (examen.DescripcionExamen.contains( other: "<p>") && examen.DescripcionExamen.contains( other: "</p>")){
            view.findViewById<TextView>(R.id.descripcionModal).text = descripcionSinEtiquetas
        }else{
            view.findViewById<TextView>(R.id.descripcionModal).text = examen.DescripcionExamen
        }

        val examenUsuarios=examenUsuario
        val calificacion=examenUsuario.TotalCalificacion
```

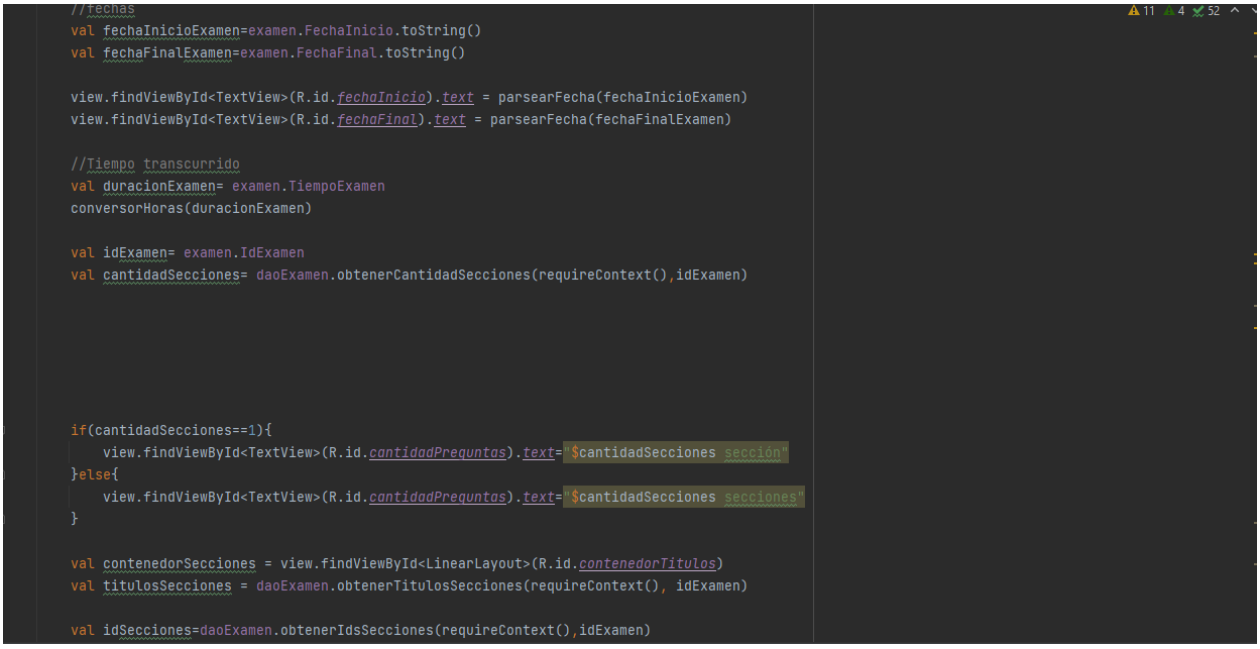
```
        if(calificacion>=60){
            view.findViewById<TextView>(R.id.tituloCalif).text="Has aprobado el examen con "
            view.findViewById<TextView>(R.id.calificacion).text="$calificacion"
            view.findViewById<TextView>(R.id.calificacion).setTextColor(ContextCompat.getColor(requireContext(), R.color.green))
        }else{
            view.findViewById<TextView>(R.id.tituloCalif).text="Has reprobado el examen con "
            view.findViewById<TextView>(R.id.calificacion).text="$calificacion"
            view.findViewById<TextView>(R.id.calificacion).setTextColor(ContextCompat.getColor(requireContext(), R.color.red))
        }
        val fechaFin=examenUsuario.TiempoExamenFinal.toString()
        view.findViewById<TextView>(R.id.fechaFinalizacion).text=parsearFecha(fechaFin)

        val tiempoTranscurrido=examenUsuario.TiempoTranscurrido
        val tiempoConvertido=conversorHoras(tiempoTranscurrido)

        view.findViewById<TextView>(R.id.tiempoTranscurrido).text=tiempoConvertido


        builder.setView(view)
            .setTitle("Detalles del examen")
            .setPositiveButton( text: "Cerrar") { dialog, _ ->
                dialog.dismiss()
            }

        return builder.create()
    }
```

```
    Edwin Canul
    fun conversorHoras(duracionExam: Int): String {
        if(duracionExam>=60) {
            val duracionHoras = duracionExam / 60
            return "$duracionHoras horas"
        }else {
            return "$duracionExam minutos"
        }
    }


    Edwin Canul
    fun parsearFecha(fechas: String): String {
        val dateFormat = SimpleDateFormat( pattern: "yyyy-MM-dd HH:mm:ss", Locale.US)
        val parsedDate = dateFormat.parse(fechas)
        val formattedDate = SimpleDateFormat( pattern: "yyyy-MM-dd hh:mm:ss a", Locale.US).format(parsedDate)

        return formattedDate.toString()
    }
}
```

No iniciado:



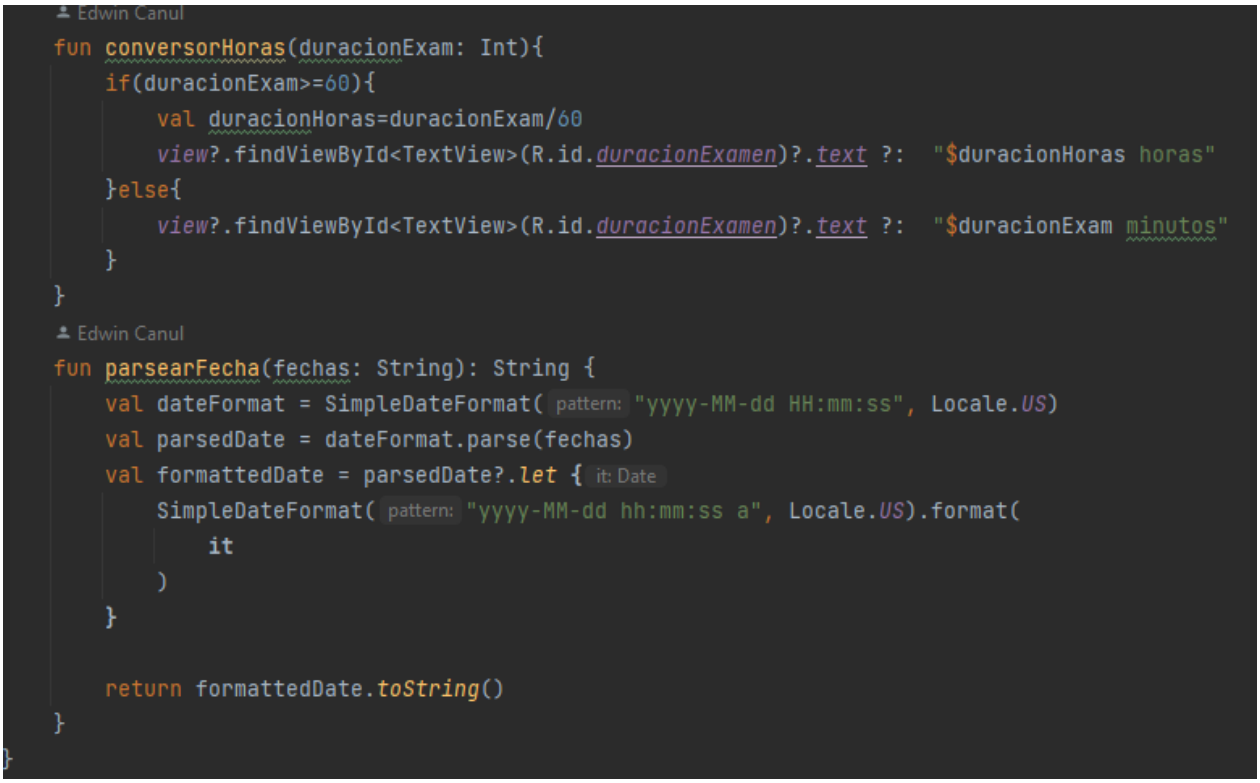
```
package com.example.examenesseq.Fragments.Inicio.ExamenAdapter

import ...

Edwin Canul
class ExamenModal(private val examen: Examen) : DialogFragment() {

    val daoExamen = DaoExamen()

    private val apiServicio: ApiServicio by lazy {
        ApiServicio.create(requireContext())
    }

    Edwin Canul
    override fun onCreateDialog(savedInstanceState: Bundle?): Dialog {
        val builder = AlertDialog.Builder(requireContext())
        val inflater = requireActivity().layoutInflater
        val view = inflater.inflate(R.layout.modal_examen, root: null)


        view.findViewById<TextView>(R.id.tituloModal).text = examen.TituloExamen
        val descripcionSinEtiquetas=
            examen.DescripcionExamen.replace("<p>".toRegex(), replacement: "").replace("</p>".toRegex(), replacement: "")

        if (examen.DescripcionExamen.contains( other: "<p>") && examen.DescripcionExamen.contains( other: "</p>")){
            view.findViewById<TextView>(R.id.descripcionModal).text = descripcionSinEtiquetas
        }else{
            view.findViewById<TextView>(R.id.descripcionModal).text = examen.DescripcionExamen
        }
```



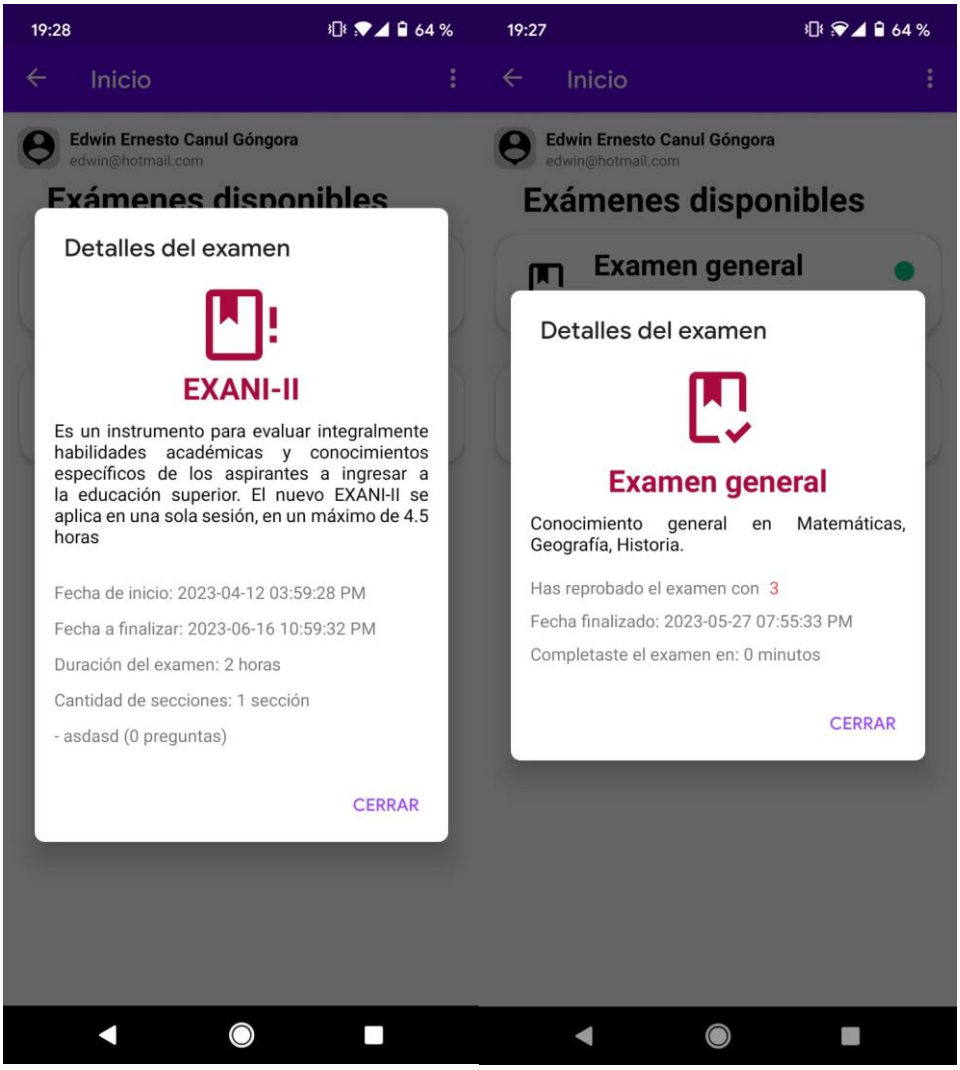
```
    //fechas
    val fechaInicioExamen=examen.FechaInicio.toString()
    val fechaFinalExamen=examen.FechaFinal.toString()

    view.findViewById<TextView>(R.id.fechaInicio).text = parsearFecha(fechaInicioExamen)
    view.findViewById<TextView>(R.id.fechaFinal).text = parsearFecha(fechaFinalExamen)

    //Tiempo transcurrido
    val duracionExamen= examen.TiempoExamen
    conversorHoras(duracionExamen)

    val idExamen= examen.IdExamen
    val cantidadSecciones= daoExamen.obtenerCantidadSecciones(requireContext(),idExamen)




    if(cantidadSecciones==1){
        view.findViewById<TextView>(R.id.cantidadPreguntas).text="$cantidadSecciones sección"
    }else{
        view.findViewById<TextView>(R.id.cantidadPreguntas).text="$cantidadSecciones secciones"
    }

    val contenedorSecciones = view.findViewById<LinearLayout>(R.id.contenedorTitulos)
    val titulosSecciones = daoExamen.obtenerTitulosSecciones(requireContext(), idExamen)

    val idSecciones=daoExamen.obtenerIdsSecciones(requireContext(),idExamen)
```

```
    for (i in 0 ≤ until < titulosSecciones.size) {
        val titulo = titulosSecciones[i]
        val idSeccion = idSecciones[i]

        val textView = TextView(requireContext())
        textView.text = "- $titulo"  // Agregar un guion antes del título
        obtenerTotalPreguntas(idSeccion) { totalPreguntas ->
            val texto = "- $titulo ($totalPreguntas preguntas)"
            textView.text = texto
        }

        contenedorSecciones.addView(textView)
    }

    builder.setView(view)
        .setTitle("Detalles del examen")
        .setPositiveButton( text: "Cerrar") { dialog, _ ->
            dialog.dismiss()
        }

    return builder.create()
}
```

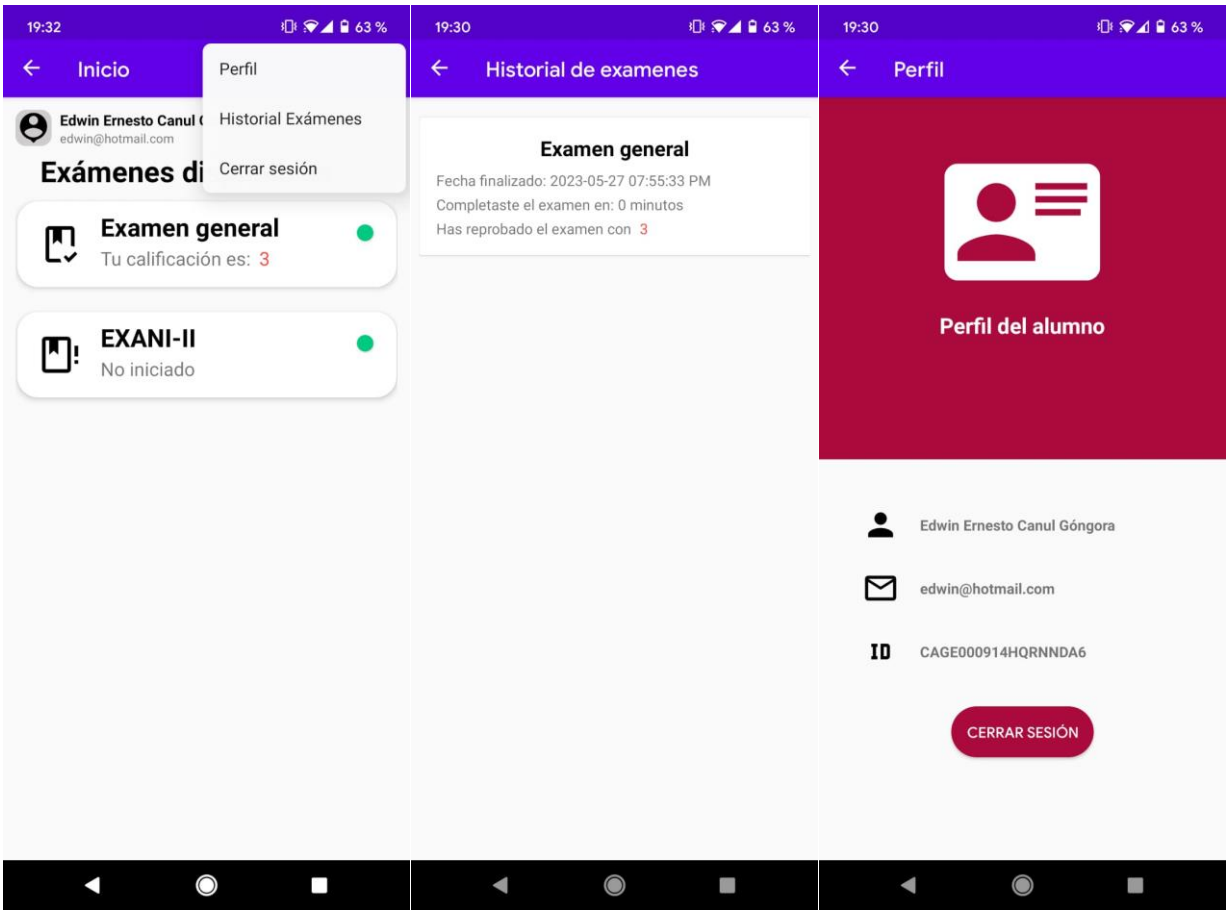
```
Edwin Canul
private fun obtenerTotalPreguntas(idSeccion: Int, callback: (Int) -> Unit) {
    apiServicio.obtenerTotalPreguntas(idSeccion).enqueue(object : Callback<LoginRespuesta> {
        override fun onResponse(call: Call<LoginRespuesta>, response: Response<LoginRespuesta>) {
            if (response.isSuccessful) {
                val loginRespuesta = response.body()
                val totalPreguntas = loginRespuesta?.Mensaje
                if (totalPreguntas != null) {
                    val preferences = PreferenceHelper.defaultPrefs(requireContext())
                    preferences.saveCantidadPreguntas(totalPreguntas)
                    callback(totalPreguntas.toInt())
                }
            } else {
                Toast.makeText(requireContext(), text: "Hubo un error en la respuesta del servidor", Toast.LENGTH_SHORT).show()
            }
        }

        override fun onFailure(call: Call<LoginRespuesta>, t: Throwable) {
            Toast.makeText(requireContext(), text: "Fallo: ${t.message}", Toast.LENGTH_SHORT).show()
            Log.e( tag: "API Failure", msg: "Error: ${t.message}", t)
        }
    })
}
```


Edwin Canul
fun conversorHoras(duracionExam: Int){
if(duracionExam>=60){
val duracionHoras=duracionExam/60
view?.findViewById<TextView>(R.id.duracionExamen)?.text ?: "$duracionHoras horas"
}else{
view?.findViewById<TextView>(R.id.duracionExamen)?.text ?: "$duracionExam minutos"
}
}
Edwin Canul
fun parsearFecha(fechas: String): String {
val dateFormat = SimpleDateFormat(pattern: "yyyy-MM-dd HH:mm:ss", Locale.US)
val parsedDate = dateFormat.parse(fechas)
val formattedDate = parsedDate?.let { it: Date
SimpleDateFormat(pattern: "yyyy-MM-dd hh:mm:ss a", Locale.US).format(
it
)
}
return formattedDate.toString()
}
}

Resultados:


19:28
64 %
Inicio
Edwin Ernesto Canul Góngora
edwin@hotmail.com
Exámenes disponibles
Detalles del examen
EXANI-II
Es un instrumento para evaluar integralmente habilidades académicas y conocimientos específicos de los aspirantes a ingresar a la educación superior. El nuevo EXANI-II se aplica en una sola sesión, en un máximo de 4.5 horas
Fecha de inicio: 2023-04-12 03:59:28 PM
Fecha a finalizar: 2023-06-16 10:59:32 PM
Duración del examen: 2 horas
Cantidad de secciones: 1 sección
- asdasd (0 preguntas)
CERRAR
19:27
64 %
Inicio
Edwin Ernesto Canul Góngora
edwin@hotmail.com
Exámenes disponibles
Examen general
Detalles del examen
Examen general
Conocimiento general en Matemáticas, Geografía, Historia.
Has reprobado el examen con 3
Fecha finalizado: 2023-05-27 07:55:33 PM
Completaste el examen en: 0 minutos
CERRAR

Dentro de esta parte de inicio se agrego un menu en donde puedes acceder a tus datos generales del perfil y tu historial de examenes:

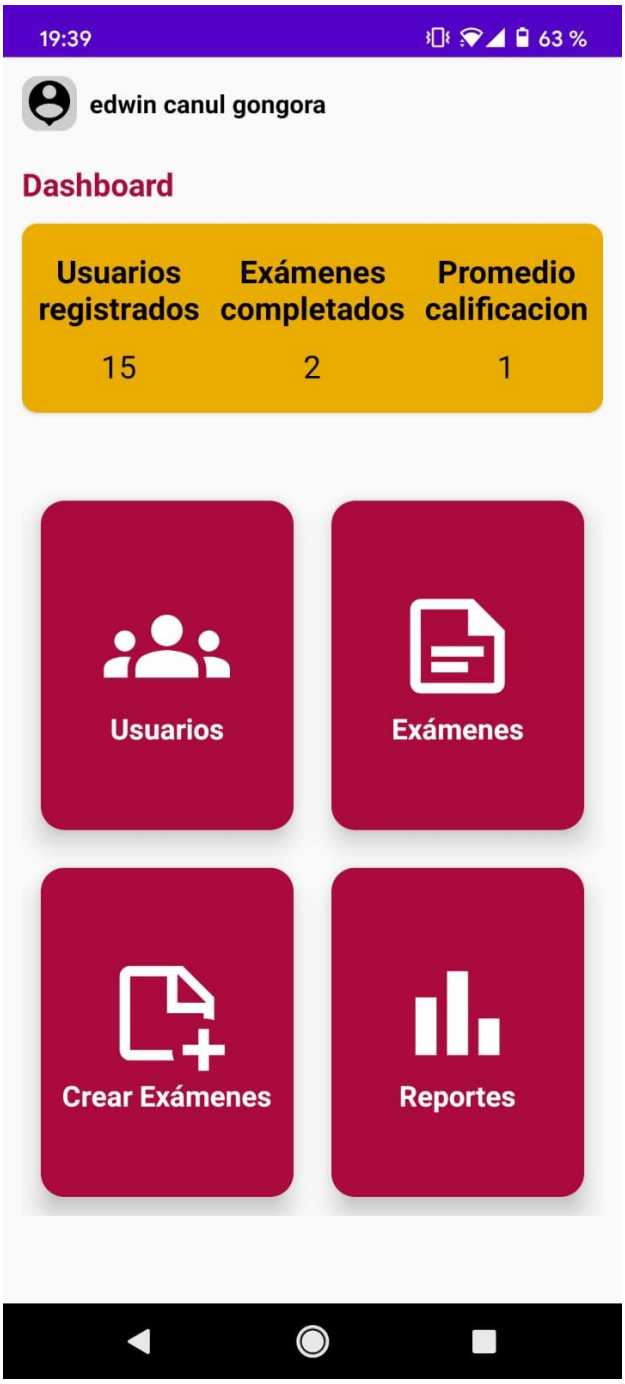


Finalmente, este tiene la funcionalidad de cerrar la sesión, el cual destruye el JSESSIONID para que este no se quede almacenado.

Esto fue todo por la parte del usuario regular.

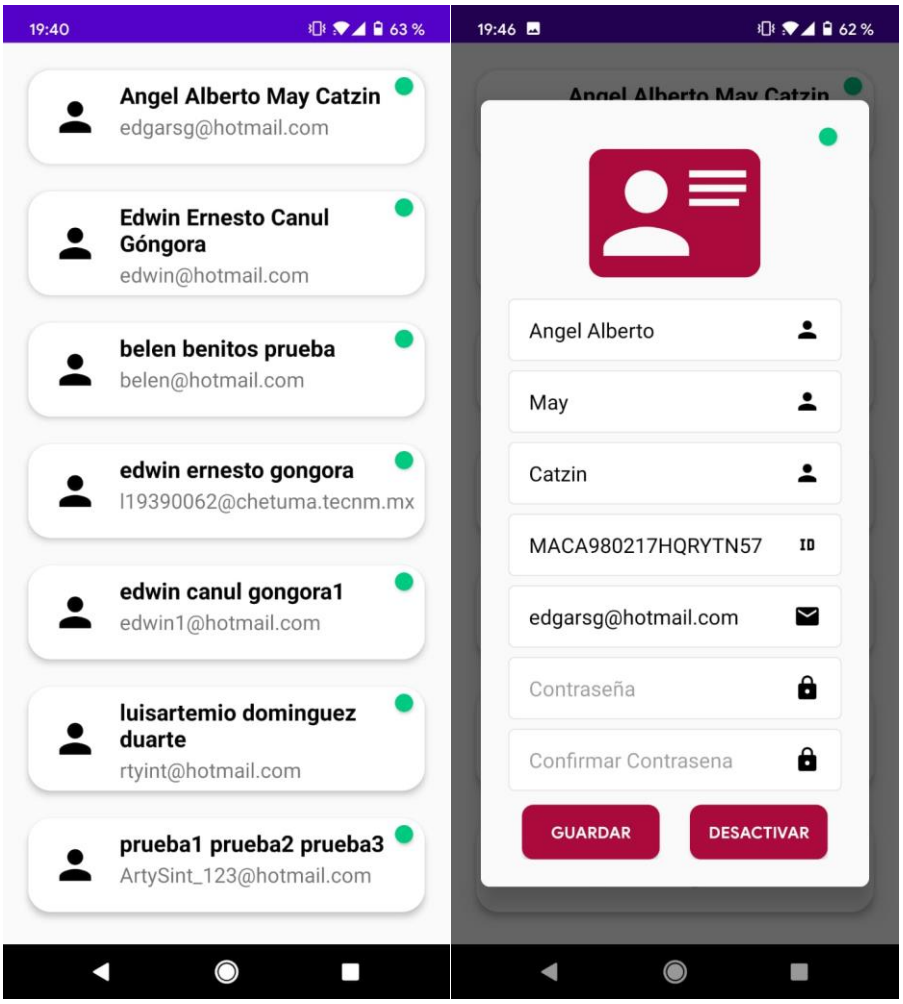
Administrativo

Para el lado administrativo se realizo un dashboard con información regular de la aplicación:

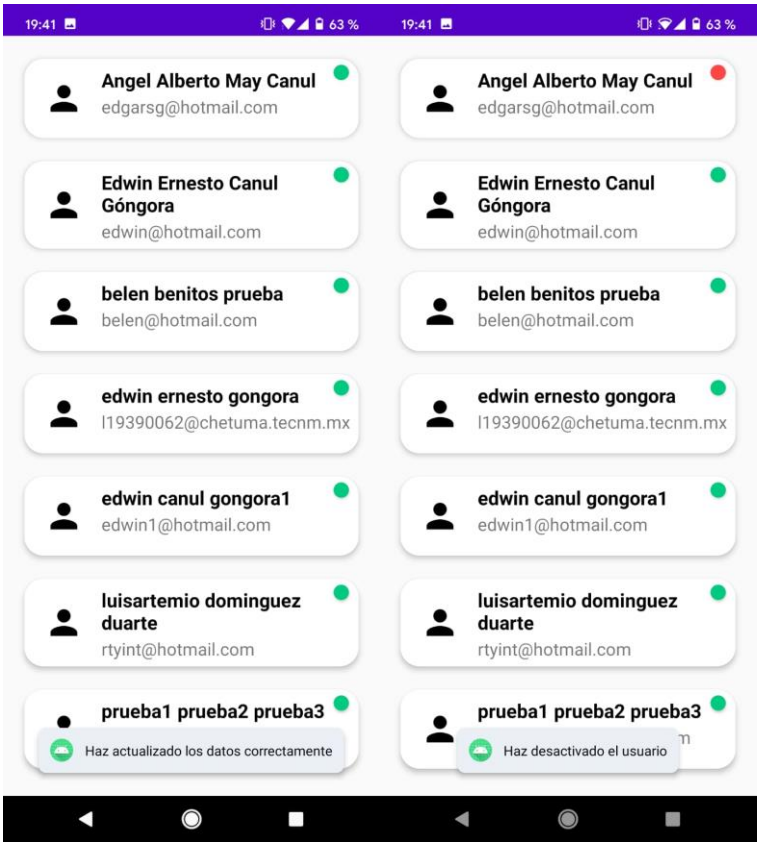


De momento solo el modulo de usuarios es el que esta en funcionamiento, el cual al darle click te lleva al siguiente fragmento:

Dentro de este igualmente puedes realizar distintas cosas, al darle click a cada itemview se despliega un modal en donde se puede modificar el usuario:



Al darle al botón guardar y desactivar actualizaba el recyclerview y cerraba el modal, mostrando los siguientes mensajes:



Y esto es todo lo que hemos realizado por la parte administrativa. El cual es simple pero la presente aplicación será presentada a SEQ para poder tener su opinión sobre hacer una aplicación móvil, ya que actualmente no se encuentra dentro de sus planes.

Credenciales de acceso

En el caso que requiera probar la app, se le dejará un apk, actualmente el proyecto lo subimos a una instancia EC2 de aws por lo que funciona correctamente con una conexión a internet.

Usuario regular:

- Correo Electrónico: edwin@hotmail.com
- Contraseña: 2

Usuario administrativo:

- Correo Electrónico: edwin1
- Contraseña: 3