BILLFACTU

Índice

Descripción del proyecto y ámbito de implantación

He desarrollado una aplicación la cual permite a las empresas crear su propio espacio en el cual enviar a sus clientes las facturas y prefecturas. He creado esto ya que en mi empresa en la cual realice mi FCT vi que esto lo hacía una persona manualmente enviando correos electrónicos. También, esto traía problemas porqué había gente que no le llegaban los correos o llamaba para pedir que le enviaran alguna factura. Como tecnologías he decidido usar django para la parte del servidor en conjunto a firebase. La parte de django está pensada para interactuar directamente con la aplicación mediante endpoints permitiendo el registro, el envío de datos, etc. Firebase lo implemente para poder enviar notificaciones a los móviles cuando se les sube facturas y prefecturas. Para la parte del cliente he usado android studio a la hora de desarrollar la aplicación. En un futuro me gustaría migrar el servidor a un servicio de cloud y la aplicación a la play store. También darle un enfoque a la aplicación de código abierto y que la gente pueda desarrollar y proponer mejoras con el fin de obtener más solidez y estabilidad en la aplicación.

Temporalización del proyecto y fases de desarrollo

La estructura de desarrollo del proyecto está dividida en varias partes:

- Preparación e idea inicial del proyecto: Un pequeño proceso en el cual se piensa la idea del proyecto y se estructura los procesos que se quieren llevar a cabo.
- Prediseño de la aplicación: Es el proceso de diseñar bocetos acerca de la aplicación, generando una idea de la estructura de la aplicación y de ahí partir con el desarrollo del backend y del frontend adaptándose a necesidades posteriores.
- Prediseño de la estructura del api rest: Es el proceso de diseñar cómo se va a estructurar el api rest, la estructura de los modelos y endpoints de esta basándose en las necesidades que parten de el prediseño de la aplicación.
- Creación backend aplicación: Desarrollo completo del proyecto DJango, adaptándose a las necesidades y contratiempos durante el desarrollo del mismo a la par del frontend.
- Creación frontend aplicación: Desarrollo completo de la aplicación android. Adaptándose a las limitaciones de esta.

	Fecha de inicio	**Fecha de fin**	**Precedente**	**Tiempo en días**
Desarrollo del proyecto	**11/04/2024**	**-**	**-**	**-**
A - Preparación y idea inicial del proyecto	**11/04/2024**	**25/04/2024**	**-**	**14**
B - Prediseño de la aplicación	**20/04/2024**	**25/04/2024**	**-**	**5**
C - Prediseño de la estructura del api rest	**23/04/2024**	**25/04/2024**	**-**	**2**
D - Creación backend aplicación	**25/04/2024**		**C**	
E- Creación frontend aplicación	**25/04/2024**		**C**	

Recursos de hardware y software

Requisitos por parte de la aplicación android:

Requisitos mínimos:

Hardware:

- Procesador: Al menos 1 GHz.
- Memoria RAM: Al menos 1 GB de RAM.
- Espacio en disco: 4MB para la instalación de la aplicación y espacio adicional para almacenar facturas descargadas.

Software:

- Sistema operativo: Android 9.0 (Pie) o superior.
- API de Android: API nivel 28 o superior.
- Bibliotecas y frameworks: La aplicación utiliza Google Analytics y Google Crashlytics, por lo que es necesario tener acceso a Internet para su funcionamiento.

Requisitos recomendados:

Hardware:

- Procesador: Al menos 1.5 GHz.
- Memoria RAM: Al menos 2 GB de RAM.
- Espacio en disco: 40 MB.

Software:

- Sistema operativo: Android 12.0 o superior.
- API de Android: Android API nivel 31 o superior.
- Bibliotecas y frameworks: La aplicación utiliza Google Analytics y Google Crashlytics, por lo que es necesario tener acceso a Internet para su funcionamiento.

Requisitos por parte del servidor DJango.

Requisitos mínimos:

Hardware:

- Procesador: Al menos 1 un núcleo a 0.5GHz
- Memoria RAM: Al menos 1 GB de RAM.
- Espacio en disco: 100 MB + Espacio ocupado por las facturas y prefacturas

Software:

- Sistema operativo: Windows 7, Linux o macOS 10.12 (Sierra) .
- Versión de Python: Python 3.6 o superior.
- Framework Django: Versión compatible con Python 3.6.
- Base de datos: Cualquier motor de base de datos compatible con Django (SQLite, PostgreSQL, MySQL, Oracle o SQL Server).
- Dependencias adicionales: Debes tener instalado el modulo bcrypt.

Requisitos recomendados:

Hardware:

- Procesador: Al menos dos núcleos a 1GHz
- Memoria RAM: Se recomienda al menos 2 GB de RAM para un rendimiento óptimo
- Espacio en disco: 200 MB + Espacio ocupado por las facturas y prefacturas

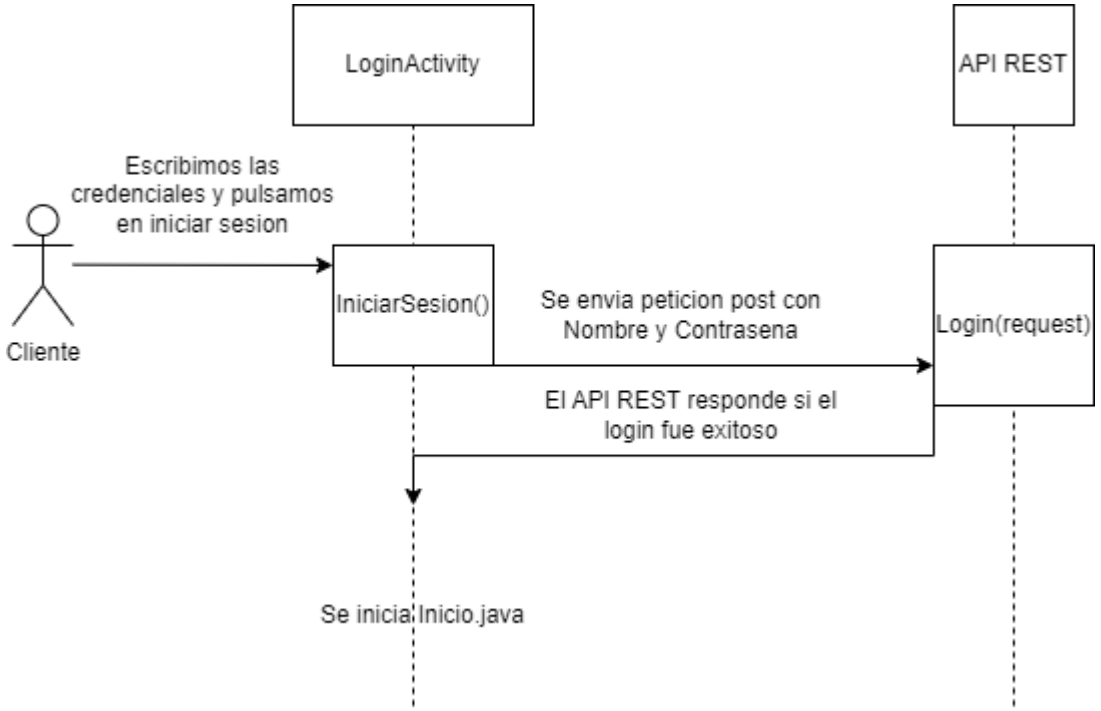
Software:

- Sistema operativo: Windows 10, Linux o 11 Big Sur.
- Versión de Python: Django es compatible con Python 3.12 o superior.
- Framework Django: Se requiere tener Django instalado.
- Base de datos: Cualquier motor de base de datos compatible con Django (SQLite, PostgreSQL, MySQL, Oracle o SQL Server).
- Dependencias adicionales: Módulo bcrypt

Arquitectura software y de sistemas

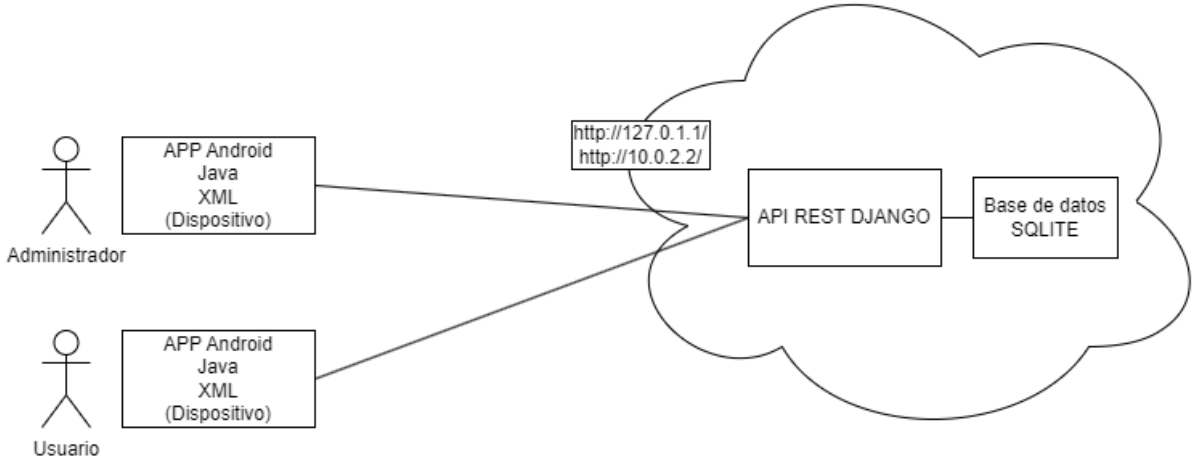
Arquitectura software:

Inicio de sesión:



Arquitectura hardware:

Estructura app/servidor:



Descripción de datos

Endpoint login DJango:

```
@csrf_exempt
def login(request):
    if request.method == 'POST':
        body_json = json.loads(request.body)
        if 'nombre' not in body_json or 'contrasena' not in body_json:
            return JsonResponse({'error': 'Faltan parámetros o parámetros incorrectos'},
status=400)

        json_nombre = body_json['nombre']
        json_password = body_json['contrasena']

        try:
            usuario = Usuario.objects.get(nombre=json_nombre)
        except Usuario.DoesNotExist:
            try:
                usuario = Usuario.objects.get(correo=json_nombre)
            except Usuario.DoesNotExist:
                return JsonResponse({'error': 'Usuario no encontrado'}, status=404)

        if bcrypt.checkpw(json_password.encode('utf-8'), usuario.contrasena.encode('utf-8')):
            return JsonResponse({'token': usuario.token, 'jefe': usuario.jefe})
        else:
            return JsonResponse({'error': 'Contraseña incorrecta'}, status=400)
```

Este fragmento describe la función login, que maneja las solicitudes de inicio de sesión. Se espera una solicitud POST con datos JSON que incluyan un nombre de usuario (nombre) y una contraseña (contrasena).Se verifica si el usuario existe en la base de datos, primero buscando por nombre y luego por correo. Si el usuario existe, se verifica la contraseña utilizando el método bcrypt.checkpw. Si la contraseña es correcta, se devuelve un token de sesión junto con un indicador de si el usuario es un jefe (jefe).

Endpoint register DJango

```
@csrf_exempt
def register(request):
    if request.method == 'POST':
        body_json = json.loads(request.body)
        if 'nombre' not in body_json or 'correo' not in body_json or 'contrasena' not in body_json
or 'jefe' not in body_json:
            return JsonResponse({'error': 'Faltan parámetros o parámetros incorrectos'},
status=400)

        json_nombre = body_json['nombre']
        json_correo = body_json['correo']
```

```
        json_password = body_json['contrasena']
        json_jefe = body_json['jefe']

        if Usuario.objects.filter(nombre=json_nombre).exists() or
Usuario.objects.filter(correo=json_correo).exists():
            return JsonResponse({'error': 'El usuario o el correo ya existe'}, status=400)

        salt = bcrypt.gensalt()
        hashed = bcrypt.hashpw(json_password.encode('utf-8'), salt)

        token = secrets.token_hex(10)

        usuario = Usuario(nombre=json_nombre, correo=json_correo,
contrasena=hashed.decode('utf-8'), token=token, jefe=json_jefe)
        usuario.save()

        return JsonResponse({'token': usuario.token})
    else:
        return JsonResponse({'error': 'Método no permitido'}, status=405)
```

Este fragmento describe la función register, que maneja las solicitudes de registro de usuarios. Se espera una solicitud POST con datos JSON que incluyan un nombre de usuario (nombre), correo electrónico (correo), contraseña (contrasena) y un indicador de si el usuario es un jefe (jefe). Se verifica si el nombre de usuario o el correo electrónico ya existen en la base de datos. Se genera una contraseña cifrada utilizando la biblioteca bcrypt. Se crea un token único para el usuario. Se crea un nuevo objeto Usuario y se guarda en la base de datos.

Pantalla login Android:

```
private void iniciarSesion() {
    String usuario = usuarioEditText.getText().toString();
    String contrasena = contrasenaEditText.getText().toString();

    if (usuario.isEmpty() || contrasena.isEmpty()) {
        Toast.makeText(this, "Por favor, complete todos los campos",
Toast.LENGTH_SHORT).show();
        return;
    } else {
        OkHttpClient client = new OkHttpClient();
        String url = Server.URL + "login/";

        MediaType JSON = MediaType.parse("application/json; charset=utf-8");
        JSONObject jsonObject = new JSONObject();
        try {
            jsonObject.put("nombre", usuario);
            jsonObject.put("contrasena", contrasena);
        } catch (JSONException e) {
```

```
            e.printStackTrace();
        }
        RequestBody body = RequestBody.create(JSON, jsonObject.toString());

        Request request = new Request.Builder()
                .url(url)
                .post(body)
                .build();

        client.newCall(request).enqueue(new Callback() {
            @Override
            public void onFailure(Call call, IOException e) {
                e.printStackTrace();
            }

            @Override
            public void onResponse(Call call, Response response) throws IOException {
                if (response.isSuccessful()) {
                    String responseStr = response.body().string();

                    try {
                        JSONObject jsonObject = new JSONObject(responseStr);
                        String token = jsonObject.getString("token");
                        boolean esJefe = jsonObject.getBoolean("jefe");

                        SharedPreferences sharedPreferences =
getSharedPreferences("MisPreferencias", MODE_PRIVATE);
                        SharedPreferences.Editor editor = sharedPreferences.edit();
                        editor.putString("token", token);
                        editor.putBoolean("esJefe", esJefe);
                        editor.apply();

                    } catch (JSONException e) {
                        e.printStackTrace();
                    }
                } else {
                    String responseStr = response.body().string();
                    String mensajeError = "Error al conectarse con el servidor";

                    try {
                        JSONObject jsonObject = new JSONObject(responseStr);
                        mensajeError = jsonObject.getString("error");
                    } catch (JSONException e) {
                        e.printStackTrace();
                    }

                    final String finalMensajeError = mensajeError;
```

```
                runOnUiThread(new Runnable() {
                    @Override
                    public void run() {
                        Snackbar.make(findViewById(R.id.main), finalMensajeError,
Snackbar.LENGTH_LONG).show();
                    }
                });
            }
        }
    });
  }
}
```

Este método se encarga de manejar el proceso de inicio de sesión del usuario en la aplicación. Se obtienen los valores ingresados por el usuario en los campos de texto para el nombre de usuario y la contraseña. Se verifica si los campos de usuario y contraseña están vacíos. Si alguno de ellos está vacío, se muestra un mensaje de error al usuario indicando que debe completar ambos campos. Se crea una solicitud HTTP POST para enviar los datos del usuario al servidor. Se utiliza la biblioteca OkHttpClient para crear la solicitud y configurarla con la URL del servidor y los datos del usuario en formato JSON. Se envía la solicitud al servidor utilizando client.newCall(request).enqueue(...). Esto se hace de forma asíncrona para no bloquear el hilo principal de la interfaz de usuario mientras se espera la respuesta del servidor. Se implementa un Callback para manejar las respuestas del servidor. Si la respuesta es exitosa (response.isSuccessful()), se procesa la respuesta y se extrae el token de acceso y otra información relevante del usuario del cuerpo de la respuesta. Luego, estos datos se almacenan en SharedPreferences para usarlos en otras partes de la aplicación. Si la respuesta no es exitosa, se muestra un mensaje de error al usuario utilizando un Snackbar.