Registro Diario de Avances – Sprint 3 Braille Express

**Fecha:** 01/05/2025

**Autor:** Ylia Jamile Ochoa Gutierrez

**Versión:** 1.0

# 1. Introducción

Este documento detalla el seguimiento diario del Sprint 3 del proyecto Braille Express, enfocado en la implementación de funcionalidades relacionadas con el ingreso manual y por voz de texto, para mejorar la accesibilidad y usabilidad de la aplicación. En este Sprint se han planificado un total de 7 tareas, correspondientes a las dos historias de usuario principales: HI 2.1 (Ingreso manual de texto) y HI 2.2 (Ingreso de texto por voz). Se incluyen las actividades diarias, las tareas completadas, evidencias de código, un burndown chart y un análisis del rendimiento del equipo.

# 2. Lineamientos de Diseño

En este Sprint se priorizó desarrollar una primera versión funcional de las siguientes características en Braille Express:

* Validación de caracteres y signos de puntuación en el ingreso manual de texto.
* Integración de la conversión de texto a Braille para el ingreso manual.
* Diseño de la experiencia de usuario para el ingreso de texto por voz.
* Integración de la API de reconocimiento de voz para transcribir el texto hablado.
* Implementación de edición sobre el texto transcrito por voz.
* Realización de pruebas funcionales para asegurar la calidad en ambos métodos de ingreso.

# 3. Historias de Usuario y Tareas

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Backlog ID | Historia de Usuario | Tiempo Estimado | Fecha de Inicio | Fecha de Finalización |
| HU-2.1 | Ingreso manual de texto | 5 pts | 12/05/2025 | 29/04/2025 |
| HU-2.2 | Ingreso de texto por voz | 8 pts | 29/04/2025 | 19/05/2025 |

# 4. Evidencias de Código

## 4.1 Historia de Usuario 2.1 — Ingreso manual de texto

Archivo: MapaBrailleModel.java

public class MapaBrailleModel {  
  
 private final Map<Character, String> mapaBraille;  
  
 public MapaBrailleModel() {  
 mapaBraille = new HashMap<>();  
  
 // Letras (usando representación Unicode Braille ⠁ = U+2801)  
 mapaBraille.put('a', "⠁");  
 mapaBraille.put('b', "⠃");  
 mapaBraille.put('c', "⠉");  
 mapaBraille.put('d', "⠙");  
 mapaBraille.put('e', "⠑");  
 mapaBraille.put('f', "⠋");  
 mapaBraille.put('g', "⠛");  
 mapaBraille.put('h', "⠓");  
 mapaBraille.put('i', "⠊");  
 mapaBraille.put('j', "⠚");  
 mapaBraille.put('k', "⠅");  
 mapaBraille.put('l', "⠇");  
 mapaBraille.put('m', "⠍");  
 mapaBraille.put('n', "⠝");  
 mapaBraille.put('o', "⠕");  
 mapaBraille.put('p', "⠏");  
 mapaBraille.put('q', "⠟");  
 mapaBraille.put('r', "⠗");  
 mapaBraille.put('s', "⠎");  
 mapaBraille.put('t', "⠞");  
 mapaBraille.put('u', "⠥");  
 mapaBraille.put('v', "⠧");  
 mapaBraille.put('w', "⠺");  
 mapaBraille.put('x', "⠭");  
 mapaBraille.put('y', "⠽");  
 mapaBraille.put('z', "⠵");  
  
 // Números (Braille literario usual usa prefijo de número)  
 mapaBraille.put('1', "⠼⠁");  
 mapaBraille.put('2', "⠼⠃");  
 mapaBraille.put('3', "⠼⠉");  
 mapaBraille.put('4', "⠼⠙");  
 mapaBraille.put('5', "⠼⠑");  
 mapaBraille.put('6', "⠼⠋");  
 mapaBraille.put('7', "⠼⠛");  
 mapaBraille.put('8', "⠼⠓");  
 mapaBraille.put('9', "⠼⠊");  
 mapaBraille.put('0', "⠼⠚");  
  
 // Signos de puntuación comunes  
 mapaBraille.put('.', "⠲");  
 mapaBraille.put(',', "⠂");  
 mapaBraille.put(';', "⠆");  
 mapaBraille.put(':', "⠒");  
 mapaBraille.put('?', "⠦");  
 mapaBraille.put('!', "⠖");  
 mapaBraille.put('(', "⠶");  
 mapaBraille.put(')', "⠶");  
 mapaBraille.put('"', "⠦");  
 mapaBraille.put('-', "⠤");  
 mapaBraille.put('\'', "⠄");  
 mapaBraille.put('/', "⠌");  
 mapaBraille.put('@', "⠈");  
 mapaBraille.put(' ', " "); // Espacio  
 }  
  
 */\*\*  
 \* Devuelve la representación en braille de un carácter dado.  
 \*/* public String obtenerBraille(char c) {  
 c = Character.*toLowerCase*(c); // Para que sea case-insensitive  
 return mapaBraille.getOrDefault(c, "⍰"); // Caracter desconocido  
 }

Archivo: BrailleTraductorModel.java

public public class BrailleTraductorModel {  
  
 private MapaBrailleModel mapaBrailleModel;  
  
 public BrailleTraductorModel() {  
 mapaBrailleModel = new MapaBrailleModel();  
 }  
  
 *//Convierte un texto completo a su representación en Braille* public String convertirTextoABraille(String texto) {  
 StringBuilder resultado = new StringBuilder();  
  
 for (char c : texto.toCharArray()) {  
 if (Character.*isUpperCase*(c)) {  
 resultado.append("⠠"); // Prefijo de mayúscula  
 c = Character.*toLowerCase*(c); // Convertir a minúscula para obtener su símbolo Braille  
 }  
  
 String simboloBraille = mapaBrailleModel.obtenerBraille(c);  
 resultado.append(simboloBraille);  
 }  
  
 return resultado.toString();  
 }  
}

Archivo: TraductorFragment.java

public public class TraductorFragment extends Fragment {  
  
 private FragmentTraductorBinding binding;  
 private TraductorViewModel traductorViewModel;  
  
 private static final int *REQUEST\_CODE\_SPEECH\_INPUT* = 1;  
  
 @Override  
 public View onCreateView(@NonNull LayoutInflater inflater,  
 ViewGroup container, Bundle savedInstanceState) {  
  
 traductorViewModel = new ViewModelProvider(this).get(TraductorViewModel.class);  
 binding = FragmentTraductorBinding.*inflate*(inflater, container, false);  
 View root = binding.getRoot();  
  
 // Traducción automática al escribir  
 binding.txtEntrada.addTextChangedListener(new TextWatcher() {  
 @Override public void beforeTextChanged(CharSequence s, int start, int count, int after) {}  
 @Override public void onTextChanged(CharSequence s, int start, int before, int count) {  
 traductorViewModel.setTextoOriginal(s.toString());  
 }  
 @Override public void afterTextChanged(Editable s) {}  
 });  
  
 // Observa el resultado traducido  
 traductorViewModel.getTextoBraille().observe(getViewLifecycleOwner(), braille -> {  
 binding.txtSalida.setText(braille);  
 });  
  
 // 🎤 Botón de micrófono  
 binding.btnHablar.setOnClickListener(v -> iniciarReconocimientoVoz());  
  
 return root;  
 }  
  
 private void iniciarReconocimientoVoz() {  
 Intent intent = new Intent(RecognizerIntent.*ACTION\_RECOGNIZE\_SPEECH*);  
 intent.putExtra(RecognizerIntent.*EXTRA\_LANGUAGE\_MODEL*,  
 RecognizerIntent.*LANGUAGE\_MODEL\_FREE\_FORM*);  
 intent.putExtra(RecognizerIntent.*EXTRA\_LANGUAGE*, Locale.*getDefault*());  
 intent.putExtra(RecognizerIntent.*EXTRA\_PROMPT*, "Habla ahora...");  
  
 try {  
 startActivityForResult(intent, *REQUEST\_CODE\_SPEECH\_INPUT*);  
 } catch (Exception e) {  
 Toast.*makeText*(getContext(), "Error al iniciar el reconocimiento de voz", Toast.*LENGTH\_SHORT*).show();  
 }  
 }  
  
 @Override  
 public void onActivityResult(int requestCode, int resultCode,  
 @Nullable Intent data) {  
 super.onActivityResult(requestCode, resultCode, data);  
 if (requestCode == *REQUEST\_CODE\_SPEECH\_INPUT* && resultCode == Activity.*RESULT\_OK* && data != null) {  
 ArrayList<String> resultados = data.getStringArrayListExtra(RecognizerIntent.*EXTRA\_RESULTS*);  
 if (resultados != null && resultados.size() > 0) {  
 String textoReconocido = resultados.get(0);  
 binding.txtEntrada.setText(textoReconocido); // Esto activa automáticamente la traducción  
 }  
 }  
 }  
  
 @Override  
 public void onDestroyView() {  
 super.onDestroyView();  
 binding = null;  
 }  
}

Archivo: TraductorViewModel.java

public class TraductorViewModel extends ViewModel {  
  
 // LiveData para almacenar y observar el texto original ingresado por el usuario.  
 private final MutableLiveData<String> textoOriginal = new MutableLiveData<>();  
  
 // LiveData para almacenar y observar el texto traducido a Braille.  
 private final MutableLiveData<String> textoBraille = new MutableLiveData<>();  
  
 // Instancia del modelo que contiene la lógica de conversión a Braille.  
 private final BrailleTraductorModel brailleTraductorModel = new BrailleTraductorModel();  
  
 // Metodo público para establecer el texto original desde la vista.  
 public void setTextoOriginal(String texto) {  
 textoOriginal.setValue(texto); // Actualiza el texto original  
 traducirTextoABraille(texto); // Traduce el texto al formato Braille  
 }  
  
 // Metodo para exponer el texto original como LiveData para ser observado por la UI.  
 public LiveData<String> getTextoOriginal() {  
 return textoOriginal;  
 }  
  
 // Metodo para exponer el texto traducido a Braille como LiveData.  
 public LiveData<String> getTextoBraille() {  
 return textoBraille;  
 }  
  
 // Metodo privado que realiza la conversión del texto original a Braille.  
 private void traducirTextoABraille(String texto) {  
 String resultado = brailleTraductorModel.convertirTextoABraille(texto); // Llama al modelo para convertir  
 textoBraille.setValue(resultado); // Actualiza el LiveData con el resultado traducido  
 }  
}

}

Archivo: fragment\_traductor.xml

<!-- Estado Bluetooth + Switch -->  
 <TextView  
 android:id="@+id/txtEstadoBluetooth"  
 android:layout\_width="0dp"  
 android:layout\_height="wrap\_content"  
 android:text="Estado: Desconectado"  
 android:textColor="#FF0303"  
 android:textSize="16sp"  
 android:textStyle="bold"  
 app:layout\_constraintStart\_toStartOf="parent"  
 app:layout\_constraintTop\_toTopOf="parent"  
 app:layout\_constraintEnd\_toStartOf="@+id/swBluetooth"  
 app:layout\_constraintHorizontal\_bias="0"/>  
  
 <Switch  
 android:id="@+id/swBluetooth"  
 android:layout\_width="wrap\_content"  
 android:layout\_height="wrap\_content"  
 android:onClick="abrirConfiguracionBluetooth"  
 app:layout\_constraintTop\_toTopOf="@id/txtEstadoBluetooth"  
 app:layout\_constraintEnd\_toEndOf="parent" />  
  
 <!-- Caja de texto de entrada -->  
 <EditText  
 android:id="@+id/txtEntrada"  
 android:layout\_width="367dp"  
 android:layout\_height="232dp"  
 android:layout\_marginTop="16dp"  
 android:background="@android:drawable/edit\_text"  
 android:gravity="start|top"  
 android:hint="Escribe el texto aquí"  
 android:inputType="textMultiLine"  
 android:padding="10dp"  
 android:scrollbars="vertical"  
 android:textSize="16sp"  
 app:layout\_constraintEnd\_toEndOf="parent"  
 app:layout\_constraintStart\_toStartOf="parent"  
 app:layout\_constraintTop\_toBottomOf="@id/txtEstadoBluetooth" />  
  
 <!-- Botones: Pegar - Micrófono - Copiar -->  
 <LinearLayout  
 android:id="@+id/layoutBotonesTexto"  
 android:layout\_width="0dp"  
 android:layout\_height="wrap\_content"  
 android:orientation="horizontal"  
 android:gravity="center"  
 app:layout\_constraintTop\_toBottomOf="@id/txtEntrada"  
 app:layout\_constraintStart\_toStartOf="parent"  
 app:layout\_constraintEnd\_toEndOf="parent"  
 android:layout\_marginTop="12dp">  
  
 <ImageButton  
 android:id="@+id/btnPegar"  
 android:layout\_width="70dp"  
 android:layout\_height="64dp"  
 android:background="@null"  
 android:contentDescription="Pegar texto"  
 android:src="@drawable/ic\_pegar" />  
  
 <ImageButton  
 android:id="@+id/btnHablar"  
 android:layout\_width="85dp"  
 android:layout\_height="match\_parent"  
 android:background="@null"  
 android:contentDescription="Hablar"  
 android:src="@drawable/ic\_micro" />  
  
 <ImageButton  
 android:id="@+id/btnCopiar"  
 android:layout\_width="67dp"  
 android:layout\_height="64dp"  
 android:background="@null"  
 android:contentDescription="Copiar texto"  
 android:src="@drawable/ic\_copiar" />  
 </LinearLayout>  
  
 <!-- Castellano - Traducir - Braille -->  
 <LinearLayout  
 android:id="@+id/layoutTraductores"  
 android:layout\_width="0dp"  
 android:layout\_height="wrap\_content"  
 android:orientation="horizontal"  
 android:gravity="center"  
 android:layout\_marginTop="12dp"  
 app:layout\_constraintTop\_toBottomOf="@id/layoutBotonesTexto"  
 app:layout\_constraintStart\_toStartOf="parent"  
 app:layout\_constraintEnd\_toEndOf="parent">  
  
 </LinearLayout>  
  
 <!-- Resultado en Braille -->  
 <TextView  
 android:id="@+id/txtSalida"  
 android:layout\_width="368dp"  
 android:layout\_height="234dp"  
 android:layout\_marginTop="12dp"  
 android:background="#EEEEEE"  
 android:gravity="start|top"  
 android:hint="Traducción Braille"  
 android:padding="10dp"  
 android:scrollbars="vertical"  
 android:textColor="@color/black"  
 android:textSize="16sp"  
 app:layout\_constraintEnd\_toEndOf="parent"  
 app:layout\_constraintStart\_toStartOf="parent"  
 app:layout\_constraintTop\_toBottomOf="@id/layoutTraductores" />  
  
 <!-- Botón Enviar -->  
 <Button  
 android:id="@+id/btnEnviar"  
 android:layout\_width="0dp"  
 android:layout\_height="wrap\_content"  
 android:text="Enviar"  
 android:backgroundTint="@color/purple\_500"  
 android:textColor="@color/white"  
 app:layout\_constraintTop\_toBottomOf="@id/txtSalida"  
 app:layout\_constraintStart\_toStartOf="parent"  
 app:layout\_constraintEnd\_toEndOf="parent"  
 android:layout\_marginTop="16dp" />  
</androidx.constraintlayout.widget.ConstraintLayout>

## 4.2 Historia de Usuario 2.2 — Ingreso de Texto por voz

Archivo: TraductorFragment.xml

// Botón de micrófono  
 binding.btnHablar.setOnClickListener(v -> iniciarReconocimientoVoz());  
  
 return root;  
}  
  
private void iniciarReconocimientoVoz() {  
 Intent intent = new Intent(RecognizerIntent.*ACTION\_RECOGNIZE\_SPEECH*);  
 intent.putExtra(RecognizerIntent.*EXTRA\_LANGUAGE\_MODEL*,  
 RecognizerIntent.*LANGUAGE\_MODEL\_FREE\_FORM*);  
 intent.putExtra(RecognizerIntent.*EXTRA\_LANGUAGE*, Locale.*getDefault*());  
 intent.putExtra(RecognizerIntent.*EXTRA\_PROMPT*, "Habla ahora...");  
  
 try {  
 startActivityForResult(intent, *REQUEST\_CODE\_SPEECH\_INPUT*);  
 } catch (Exception e) {  
 Toast.*makeText*(getContext(), "Error al iniciar el reconocimiento de voz", Toast.*LENGTH\_SHORT*).show();  
 }  
}

# 5. Burndown Chart

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Backlog ID** | **Historia de Usuario** | | **Tiempo Estimado** | | **12/05/2025** | | **13/05/2025** | | **14/05/2025** | | **15/05/2025** | | **16/05/2025** | | **17/05/2025** | | **19/05/2025** | |
| **Dia1** | | **Dia2** | | **Dia3** | | **Dia4** | | **Dia5** | | **Dia6** | | **Dia7** | |
| **HU-2.1** | Ingreso manual de texto | | 36 | | 10 | | 7 | | 6 | | 5 | | 6 | | 2 | | 0 | |
| **HU-2.2** | Ingreso de texto por voz | | 21 | | 1 | | 2 | | 3 | | 5 | | 2 | | 5 | | 3 | |
| **Tiempo de trabajo** | | 57 | | 11 | | 9 | | 9 | | 10 | | 8 | | 7 | | 3 | |

# 6. Análisis del Sprint

* Se completaron todas las tareas planificadas para este Sprint 3, cumpliendo el objetivo de 68 horas de trabajo.
* El equipo mantuvo un ritmo constante y eficiente, sin bloqueos significativos.
* La integración de la API de reconocimiento de voz se realizó con éxito, mejorando notablemente la interacción del usuario.
* La validación de texto ingresado manualmente fue robusta, incluyendo soporte para caracteres y signos de puntuación esenciales.
* Las pruebas funcionales demostraron la estabilidad de las nuevas funcionalidades, con ajustes menores realizados durante el Sprint.
* La modularización del código se mantuvo, facilitando la escalabilidad y mantenibilidad.

# 7. Conclusiones

El Sprint 3 de Braille Express concluyó satisfactoriamente con la implementación de funcionalidades claves para el ingreso manual y por voz de texto. Se alcanzaron los objetivos planteados dentro del tiempo estimado, cumpliendo con la planificación de 7 tareas distribuidas entre las historias de usuario HI 2.1 y HI 2.2.

El equipo demostró habilidades técnicas sólidas al integrar tecnologías de validación y reconocimiento de voz, asegurando una experiencia de usuario accesible y eficiente. La arquitectura modular y la documentación continua facilitaron la integración de estas nuevas características, sentando las bases para futuras mejoras, como la integración avanzada con servicios de accesibilidad y optimización del procesamiento de texto.

La gestión ágil del Sprint permitió resolver desafíos menores sin comprometer la productividad ni la calidad, reflejando un equilibrio óptimo en el proceso de desarrollo.