Implementación ambiente de desarrollo

Construye++

App / Panel Admin

Preparado por: José Barboza
Para: Bravo Izquierdo
Fecha: 17/11/2024

Índice

1 Introducción

Nombre	Construye++
Tipo	App Chatbot y Panel de Administrador
Sistema	Android / iOS (app) Web (panel)
Descripción	APP Chatbot para el área de la construcción Chilena y panel de administrador versión web.

Propósito del documento
Documento sobre la Implementación del ambiente de desarrollo de un **Chatbot con un panel administrativo** para el área de la construcción en Chile, en este se describe el entorno técnico necesario para que otros desarrolladores o equipos configuren su espacio de trabajo y contribuyan al proyecto.

2 Requisitos del sistema

Requisitos de Hardware (mínimos)
CPU: 2 núcleos
RAM: 8 GB
Almacenamiento: 256 GB SSD (recomendado)
Monitor: Full HD
Otros: Mouse, teclado y conexión a internet.

Requisitos de Software
SO: Windows 11, macOS Sonoma (14.5)
IDE: Visual Studio Code (2024)

Requisitos para Dispositivos
Android Versión mínima: Android 8.0 (API 26)
iOS Versión mínima: iOS 13.0
Espacio en dispositivo: 100 MB mínimo

3 Dependencias

Frameworks y SDK's		
Versión	Nombre	Descripción
^3.19.0	Flutter	Framework multiplataforma para desarrollo móvil
^34.0.0	Android SDK	Kit de desarrollo para Android
^5.0.2	Django	Framework web Python para el panel administrativo

Lenguajes de programación		
Versión	Nombre	Descripción
^3.3.0	Dart	Lenguaje principal para desarrollo Flutter, usado en la app móvil
^3.11.0	Python	Lenguaje usado para el backend con Django y Azure Functions

Librerías App		
Versión	Nombre	Descripción
^1.2.1	http	Cliente HTTP para realizar peticiones a APIs y servicios web
^2.2.3	shared_preferences	Almacenamiento persistente de datos simples clave-valor
^6.0.0	provider	Manejo de estado y dependency injection en Flutter
^1.1.2	image_picker	Permite seleccionar imágenes de la galería o cámara
^2.0.0	mixpanel_flutter	Analítica y seguimiento de eventos de usuario
^17.2.2	flutter_local_notifications	Manejo de notificaciones locales en la app
^0.5.2	workmanager	Ejecuta tareas en segundo plano de manera periódica
^3.6.0	firebase_core	Funcionalidad base de Firebase para Flutter
^15.1.3	firebase_messaging	Implementación de notificaciones push con Firebase
^11.3.3	firebase_analytics	Analítica y seguimiento de uso con Firebase

^9.2.2	flutter_secure_storage	Almacenamiento seguro de datos sensibles
^2.0.1	jwt_decoder	Decodificación y manejo de tokens JWT
^2.4.1	flutter_native_splash	Pantalla de splash nativa personalizada
^4.6.0	posthog_flutter	Analítica de producto y seguimiento de eventos
^0.19.0	intl	Internacionalización y localización
^5.0.0	flutter_lints	Reglas de análisis estático para mejor código

Librerías Panel Admin		
Versión	Nombre	Descripción
^4.2.5	Django Rest Framework	Framework para crear APIs RESTful para la comunicación con la app móvil
^3.0.1	django-cors	Manejo de CORS para permitir peticiones desde la app Flutter

Manejadores de paquetes		
Versión	Nombre	Descripción
^3.19.0	pub	Manejador de paquetes de Dart/Flutter
^24.2	pip	Manejador de paquetes de Python
^10.9	npm	Manejador de paquetes de Node.js

4 Herramientas de desarrollo

IDE/Editor de texto recomendado	
Versión	Nombre
^1.94	Visual Studio Code

Extensiones y plugins	
Versión	Nombre
^3.98	Flutter
^3.98	Dart
^2024.16	Python
^2024.12	Python Debugger
^2024.10	Pylance
^1.2	Azure Tools

Control de versiones		
Versión	Nombre	Descripción
^2.47	Git	Sistema de control de versiones distribuido para el seguimiento de cambios en el código
-	Github	Plataforma de alojamiento y colaboración de código fuente usando Git
^3.4.6	Github Desktop (opcional)	Interfaz gráfica opcional para gestionar repositorios Git locales y de Github

5 Instalación del entorno de desarrollo

Instrucciones App	
Clonar el repositorio	git clone https://github.com/Diiego0/construyeapp.git
	cd proyecto
Instalar dependencias	flutter pub get

6 Ejecución de la aplicación

Instrucciones	
Ejecutar App	flutter run
Ejecutar Panel	python manage.py runserver

7 Pruebas

Pruebas		
	Tipo	Comando
Tests App Flutter	Ejecutar tests unitarios	flutter test
	Tests de integración	flutter drive --target=test_driver/app.dart
Tests Panel Admin	Ejecutar tests Django	python manage.py test

Tests Manuales	
1. Funcionalidad App	Login/Registro
	Integración con Copilot Studio
	Notificaciones push
	Almacenamiento local
	Navegación general
2. Funcionalidad Panel	Autenticación admin
	CRUD de usuarios
	Visualización de métricas
	Endpoints API
3. Integración Azure	Conexión Azure SQL
	Azure Functions
	Azure Bot Service

Verificación pre-deployment
App construye sin errores
Panel admin inicia correctamente
APIs responden adecuadamente
Servicios Azure conectados
Tests automatizados pasan

8 Configuración del entorno de desarrollo en la nube

Configuración Nube	
Configuración de Azure Functions	
1	Crear nueva Function App en Azure Portal
2	Configurar las variables de entorno necesarias
3	Desplegar las functions usando VS Code o Azure CLI
Configuración de Azure SQL	
1	Crear servidor y base de datos en Azure Portal
2	Configurar firewall y reglas de acceso
3	Obtener connection string y configurar en .env

9 Solución de problemas comunes

Problemas	
Errores al conectarse a la base de datos (problemas de credenciales)	- Verificar string de conexión en archivo .env - Comprobar reglas de firewall de Azure SQL - Validar credenciales de acceso
Errores con dependencias (versiones incompatibles de Flutter o Django)	- Ejecutar flutter clean - Actualizar pubspec.yaml - Verificar compatibilidad de versiones
Configuración incorrecta del entorno .env (claves API o URLs faltantes)	- Crear archivo .env siguiendo ejemplo .env.example - Verificar todas las variables requeridas estén definidas - Comprobar formato de URLs y keys

10 Contribuciones

Pasos para Contribuir	
1. Planificación	
1.1	Comunicar en canales oficiales el módulo a trabajar
1.2	Esperar confirmación del equipo para evitar conflictos
1.3	Coordinar con otros desarrolladores si es necesario
2. Desarrollo	
2.1	Crear rama para el nuevo desarrollo
2.2	Seguir estándares de código del proyecto
2.3	Documentar cambios importantes
3. Pruebas	
3.1	Ejecutar tests locales
3.2	Verificar funcionalidad en app y panel
3.3	Validar integraciones con Azure y Copilot
3.4	Comprobar que no hay regresiones
4. Commit (Formato)	
4.1	[v1.2.3] Breve descripción concisa
4.2	Detalle de cambios realizados
4.3	Bugs corregidos (si aplica)
4.4	Dependencias actualizadas
5. Revisión	
5.1	Solicitar code review del equipo
5.2	Atender y resolver comentarios
5.3	Obtener aprobación antes de merge

Contribuciones Git	
1. Preparación	Fork y clone del repositorio: git clone https://github.com/[tu-usuario]/construyeapp.git
	Crear rama para nuevo feature: git checkout -b feature/[nombre-feature]
2. Durante desarrollo	Commits descriptivos: [tipo]: [descripción corta] Tipos: feat, fix, docs, style, refactor
	Mantener código actualizado: git pull origin main
3. Pull Request	Verificar cambios localmente
	Crear PR a rama main
	Descripción debe incluir: * Resumen de cambios * Screenshots (si aplica) * Tests realizados

Buenas prácticas
Commits pequeños y frecuentes
Nombres descriptivos para ramas
Testear cambios antes de PR
Mantener comunicación con el equipo