URBANTRACKER

MANUAL DE INSTALACIÓN

Sistema de gestión y visualización de rutas de transporte urbano en tiempo real

SENA

Manual de instalación

Brayan Estiven Carvajal Padilla

Diego Fernando Cuellar Hernández

Andrés Felipe Suaza Bustos

Carlos Javier Rodriguez Manchola

Aprendices

Carlos Julio Cadena

Instructor

ANÁLISIS Y DESARROLLO DE SOFTWARE

FICHA 2899747

SERVICIO NACIONAL DE APRENDIZAJE SENA

CENTRO DE LA INDUSTRIA DE LA EMPRESA Y LOS SERVICIOS

REGIONAL HUILA

2025

Contenido

Manual de Instalación

La siguiente guía detalla el proceso completo para instalar y ejecutar **UrbanTracker** en un equipo nuevo. Se cubren los requisitos previos, la configuración del entorno, la instalación de dependencias y la ejecución de cada componente del proyecto, así como soluciones a problemas comunes. *UrbanTracker* es una plataforma web en tiempo real para rastreo de buses de transporte público, compuesta por un backend Java/Spring Boot, un frontend web React (Next.js) y aplicación móvil en React Native (Expo).

1. Requisitos del sistema

Antes de comenzar, asegúrate de que el sistema cumpla con los siguientes requisitos mínimos (software base y versiones):

- **Sistema Operativo**: Windows 10/11, Linux (Ubuntu 20.04+ o equivalente) o macOS 11+ (Big Sur en adelante). Se recomienda 64 bits.
- **Java JDK**: Versión 17 o superior instalada y configurada en el PATH (requerido por Spring Boot)[1].
- **Apache Maven**: Versión 3.6+ instalada (o usar el wrapper incluido). Maven se usará para compilar y ejecutar el backend[2].
- **Node.js y NPM**: Node.js versión 16 o 18 LTS (o superior) junto con el gestor de paquetes NPM. Necesario para ejecutar las aplicaciones web y móvil (React/Expo)[2]. **Nota:** Alternativamente puede usarse Yarn, pero las instrucciones asumirán uso de NPM.
- **PostgreSQL**: Motor de base de datos PostgreSQL versión 12 o superior instalado y en ejecución localmente (por defecto en el puerto 5432)[2]. Se usará para almacenar los datos de UrbanTracker.
- **Git**: Cliente Git para clonar el repositorio desde GitHub.
- **Herramientas móviles (opcional)**: Si planeas ejecutar las aplicaciones móviles:
- **Expo CLI**: Instalado globalmente (`npm install -g expo-cli`) o usar `npx` expo. Expo facilita la ejecución de las apps React Native.
- **Entorno Android**: Android Studio con un **Emulador Android** configurado, o un dispositivo físico Android con modo desarrollador.
- **Herramientas nativas**: Si usas emulador, asegúrate de tener instalados los SDKs requeridos (por ejemplo, Android SDK nivel 33 o superior).

Además de lo anterior, se recomienda contar con una conexión a Internet (para descargar dependencias y paquetes) y al menos 4 GB de RAM libre durante la instalación/ejecución (el backend, frontend y la base de datos consumen recursos moderados).

2. Clonado del repositorio

El código fuente de UrbanTracker se encuentra en GitHub. Para obtener una copia local del repositorio, sigue estos pasos:

1. **Clonar el repositorio desde GitHub**: Abre una terminal/símbolo del sistema en la ubicación donde desees guardar el proyecto y ejecuta el comando de clonación:

```
git clone https://github.com/AFSB114/UrbanTracker.git
```

Esto descargará todos los archivos del proyecto en un directorio llamado `UrbanTracker`. A continuación, navega dentro de la carpeta del proyecto:

```
cd UrbanTracker
```

(El repositorio puede ser clonado vía HTTPS como se muestra arriba[3]. Alternativamente, si prefieres usar SSH, asegúrate de tener configurada tu clave pública y utiliza la URL SSH correspondiente.)

1. **Método alternativo (descarga manual)**: Si no tienes Git instalado, puedes descargar el código como archivo ZIP. Ve a la página del repositorio en GitHub y haz clic en **Code > Download ZIP**. Luego extrae el ZIP en la ubicación deseada.

Nota: verifica que la estructura de directorios sea correcta después de clonar/extraer. Deberías ver carpetas como `Backend`, `Web-Admin`, `Web-Client`, `Movil-Driver-Client`, etc., dentro del directorio principal del proyecto.

3. Instalación de dependencias

Con el repositorio clonado en tu equipo, procede a instalar todas las dependencias necesarias para cada componente del sistema. UrbanTracker tiene varios sub-proyectos (backend, Frontend web, app móvil), cada cual con sus propias dependencias. A continuación, se detallan los pasos para cada parte:

- **Backend (Java + Spring Boot)**: El backend utiliza Maven para la gestión de dependencias. Dirígete al directorio `Backend/` del proyecto:

```
cd UrbanTracker/Backend
```

Una vez allí, puedes usar **Maven** para descargar las dependencias y compilar el proyecto. Ejecuta:

```
mvn clean install
```

Esto descargará todos los *JARs* necesarios (framework Spring Boot, driver de PostgreSQL, etc.)[4] y compilará el código fuente. Si no tienes Maven instalado globalmente, puedes usar el *wrapper* incluido: en Linux/macOS `./mvnw clean install` o en Windows `mvnw.cmd clean install` (esto utilizará una distribución Maven empaquetada en el repo).
Consejo: La primera vez puede tardar un par de minutos en descargar todas las librerías. Si todo va bien, deberías ver un **BUILD SUCCESS** al final.

- **Base de datos (PostgreSQL)**: No hay dependencias como tal que instalar vía código, pero asegúrate de tener PostgreSQL funcionando. Si aún no lo hiciste, instala PostgreSQL desde su web oficial o usando tu gestor de paquetes. Durante la instalación, toma nota de la contraseña del usuario `postgres` (o crea un usuario específico para UrbanTracker). Más adelante configuraremos la conexión a la base de datos.

- **Frontend web (Web-Admin React)**: La aplicación web (panel de administración) está en el directorio `Web-Admin/` y fue creada con Next.js. Para instalar sus dependencias JavaScript:

- Ve al directorio del Frontend:
  `cd UrbanTracker/Web-Admin`

- Ejecuta el instalador de paquetes de Node:
  `npm install`

  Este comando leerá el archivo `package.json` del proyecto e instalará todos los paquetes listados (React, Next, librerías UI, etc.) en la carpeta `node_modules`. Asegúrate de tener conexión a Internet ya que NPM descargará los módulos necesarios. Al finalizar, deberías ver que se crearon los directorios `node_modules` y `package-lock.json` en `Web-Admin/`.

(Si prefieres Yarn, podrías usar `yarn install`, ya que el proyecto es compatible con cualquier gestor de Node. Sin embargo, asegúrate de no mezclar NPM y Yarn para evitar conflictos en `node_modules`.)

- **Aplicación móvil (React Native + Expo)**: UrbanTracker incluye una aplicación móvil (cliente de conductor) bajo el directorio `Movil-Driver-Client/`. Es un proyecto Expo/React Native separado. Debes instalar sus dependencias:
- Ve al directorio de la aplicación móvil:

```
cd UrbanTracker/Movil-Driver-Client
```

Luego ejecuta:

```
npm install
```

- Esto instalará las dependencias móviles (Expo SDK, React Native, librerías como react-native-maps, MQTT, etc.) según el `package.json`.

- *(Opcional: instalar Expo CLI)* – Si aún no lo hiciste en requisitos, es recomendable instalar la interfaz de línea de comandos de Expo de forma global:

  ```
  npm install -g expo-cli
  ```

  Aunque también puedes usar `npx expo [comando]` sin instalación global, tener la CLI facilita comandos como `expo start`. Verifica la versión con `expo --version` una vez instalado.

Tras completar estos pasos, las dependencias de todos los subproyectos deberían estar instaladas. En resumen, tendrás: - Maven descargó las librerías Java en tu caché Maven (`~/.m2`). - NPM instaló los paquetes Node para el frontend y las apps móviles en sus respectivos `node_modules`.

Si algún comando de instalación falla, consulta la sección de **Solución de errores comunes** al final de este documento. A menudo, los problemas se deben a versiones incorrectas de las herramientas o a falta de conexión a Internet.

4. Descarga de datasets necesarios

Este proyecto no requiere la descarga de datasets de terceros para funcionar, ya que opera con datos en tiempo real y los almacena en la base de datos. **No existe un conjunto de datos estático que debas descargar** para iniciar UrbanTracker. Sin embargo, sí es necesario preparar la base de datos PostgreSQL y (opcionalmente) cargar algunos datos iniciales mínimos para pruebas.

Sigue estos pasos para dejar la base de datos lista:

- **Crear la base de datos**: Inicia tu servidor PostgreSQL (si no está ya ejecutándose). Luego crea una base de datos vacía para UrbanTracker. Puedes hacerlo usando la línea de comandos de Postgres (psql) o una herramienta gráfica como PgAdmin. En psql, por ejemplo, conecta como usuario administrador (`postgres`) y ejecuta:

  ```
  CREATE DATABASE urbantracker;
  ```

Esto creará una BD llamada `urbantracker`. Puedes elegir otro nombre, pero recuerda usarlo luego en la configuración.

- **Crear usuario y asignar permisos (opcional)**: Si deseas mantener las credenciales separadas, crea un usuario dedicado:

  ```
  CREATE USER urban_user WITH PASSWORD 'tu_contraseña_segura';
  ```

  Otorga todos los permisos sobre la nueva base de datos al usuario:

  ```
  GRANT ALL PRIVILEGES ON DATABASE urbantracker TO urban_user;
  ```

  Nota: Puedes omitir este paso y usar el usuario `postgres` por simplicidad en desarrollo. En producción es recomendable un usuario limitado.

- **Datos iniciales**: UrbanTracker no viene con datos precargados de rutas o usuarios. Tras la instalación, la base de datos estará vacía. Para probar el sistema, deberás crear algunos registros manualmente:

- **Usuario administrador**: El primer paso al ejecutar podría ser crear un usuario con rol administrador para acceder al panel de administración. Dado que el backend expone endpoints REST, puedes crear usuarios vía API (por ejemplo, llamando al endpoint de registro) o insertarlos directamente en la tabla correspondiente. Si optas por la vía rápida antes de tener la API funcionando, puedes insertar un registro en la tabla `users` de PostgreSQL con un nombre de usuario y contraseña (la contraseña debe ir cifrada con BCrypt para que el login funcione, según la configuración de seguridad). Alternativamente, podrías modificar temporalmente el código para insertar un admin por defecto. Sin embargo, la manera típica es utilizar un servicio de registro expuesto por la aplicación una vez que esté en marcha.
- **Otros datos (rutas, vehículos, etc.)**: Estos se gestionan a través del panel de administración de UrbanTracker. No necesitas cargar datos de ejemplo obligatoriamente; puedes crearlos usando la propia aplicación una vez funcionando (por ejemplo, agregando rutas, conductores y vehículos mediante la interfaz administrativa).

En resumen, **no hay datasets externos que descargar**. La preparación de datos se reduce a configurar la base de datos vacía y asegurarse de tener credenciales de acceso. Una vez que ejecutes el backend, este debería crear automáticamente las tablas necesarias (usando JPA/Hibernate). Luego podrás ingresar datos a través de la aplicación. Si las tablas no se crean, podrías necesitar verificar la configuración de autogeneración de schema (ver sección de Configuraciones adicionales).

5. Configuraciones adicionales

Antes de poner en marcha UrbanTracker, es necesario realizar algunas configuraciones en archivos y variables de entorno para integrar correctamente los componentes:

- **Configuración de la base de datos en el backend**: El servidor Spring Boot necesita saber cómo conectarse a PostgreSQL. Normalmente esto se configura en el archivo `application.properties` o `application.yml`. En este proyecto, si no existe ya un archivo de configuración, crea uno. Ve al directorio `UrbanTracker/Backend/src/main/resources/` y crea un archivo llamado **`application.properties`** (si ya existe uno de ejemplo, puedes editarlo). Agrega las siguientes propiedades básicas:

  ```
  spring.datasource.url=jdbc:postgresql://localhost:5432/urbantracker
  spring.datasource.username=tu_usuario_bd   # ej. postgres o urban_user
  spring.datasource.password=tu_contraseña_bd
  spring.jpa.hibernate.ddl-auto=update
  spring.jpa.database-platform=org.hibernate.dialect.PostgreSQLDialect
  ```

  Ajusta `url`, `username` y `password` según el nombre de tu base de datos, usuario y contraseña configurados en PostgreSQL. La opción `ddl-auto=update` indica a Hibernate que cree o actualice automáticamente las tablas según las entidades JPA definidas, lo que facilita la inicialización del schema. (En entornos productivos podrías cambiarlo a `validate` o manejar migraciones con Flyway/Liquibase, pero para desarrollo `update` es conveniente).

Nota: En lugar de un archivo properties, también puedes configurar estas propiedades mediante **variables de entorno**. Por ejemplo, en sistemas Unix podrías exportar `SPRING_DATASOURCE_URL`, `SPRING_DATASOURCE_USERNAME` y `SPRING_DATASOURCE_PASSWORD` antes de ejecutar el backend. Usa el método que te sea más cómodo. Si optas por variables, asegúrate de exportarlas en la sesión actual o agregarlas a tu entorno permanente.

- **Configuración del frontend web**: La aplicación Next.js (Web-Admin) podría necesitar conocer la URL donde corre el backend para hacer peticiones AJAX. Revisa si existe un archivo `.env.local` o similar en `Web-Admin`. De no existir, podrías crear uno para variables del frontend. Por ejemplo: `NEXT_PUBLIC_API_URL=http://localhost:8080/api/v1` (ajustando al puerto donde el backend esté corriendo). La notación `NEXT_PUBLIC_` permite exponer esa variable al código del navegador. Si la aplicación web asume que el backend está en el mismo host (por ejemplo, podría llamar a `/api/...` directamente), quizás no necesite configuración extra. En cualquier caso, ten en cuenta que en modo desarrollo Next.js por defecto sirve el frontend en `http://localhost:3000`, así que las llamadas al API deberán apuntar a

`http://localhost:8080` donde estará el backend. Si fuera necesario, habilita CORS en el backend para permitir esas conexiones (Spring Boot Security junto con Spring Web puede bloquear peticiones desde otro origen si no se configuran correctamente los permisos; podrías añadir una configuración para permitir el origen `http://localhost:3000`).

- **Configuración de la aplicación móvil**: La app móvil utiliza mensajería en tiempo real (MQTT). Hay un par de cosas a verificar:

- *URL del servidor MQTT/API*: Si la app móvil necesita saber la dirección del servidor (por ejemplo, para conectarse vía MQTT o llamar a la API REST), busca en la configuración de la app (posiblemente en algún archivo de constantes o configuración) alguna URL. Asegúrate de que apunte a tu servidor backend. Si la aplicación y el backend corren en la misma máquina, es probable que la URL de la API sea `http://<IP_local>:8080`. Si usas un emulador Android, recuerda que `localhost` dentro del emulador no es el host de la máquina; para referirse al host local desde un emulador Android se utiliza la IP `10.0.2.2`. Por ejemplo, la app móvil debería llamar a `http://10.0.2.2:8080/api/...` si el backend está en tu máquina host. Verifica y ajusta esto para evitar problemas de conexión desde las apps móviles.
- *Configuración Expo*: Normalmente no se requieren cambios manuales en la configuración de Expo para correr en desarrollo. Sin embargo, si quieres personalizar el nombre de la app, iconos, etc., puedes revisar el archivo `app.json` en cada proyecto móvil.

- **Variables de entorno para JWT u otros secretos**: El backend utiliza JWT para autenticación (tiene la librería **jjwt** incluida). Asegúrate de configurar el secreto de firma JWT si el código lo requiere. Revisa si en el código se espera una propiedad como `jwt.secret` o similar. Si no encuentras algo explícito, es posible que se esté usando un valor por defecto o aún no se implementó completamente. En entornos reales, siempre define un secreto robusto para firmar los tokens JWT (por ejemplo, en `application.properties` agregar `application.jwt.secret=UnaFraseSecretaMuyDificilDeAdivinar` y usarla en el servicio JWT). Para desarrollo, puedes usar un valor sencillo, pero documenta que debe cambiarse en producción.

- **Otros ajustes**: Si deseas cambiar los puertos por defecto:

- El backend Spring Boot usa el puerto **8080** por defecto. Puedes cambiarlo estableciendo `server.port=XXXX` en `application.properties` o exportando `SERVER_PORT`.
- El frontend web Next.js por defecto usa **3000** en desarrollo. Para cambiarlo, puedes ejecutar `npm run dev -- -p 3001` (por ejemplo) o establecer la variable de entorno `PORT=3001` antes de arrancar.

- La app móvil en Expo no requieren puerto fijo, utilizan el Metro bundler que te da una URL (normalmente `exp://` or `http://` con un puerto dinámico). Solo asegúrate de que pueda comunicarse con el backend en 8080.

Tras configurar lo anterior, estarás listo para iniciar los componentes. Revisa una vez más las credenciales y rutas configuradas para evitar errores en la ejecución.

6. Ejecución del proyecto

Con todo instalado y configurado, procedemos a ejecutar los distintos componentes de UrbanTracker. Es recomendable iniciar primero los servicios de backend y base de datos, y luego las interfaces de usuario (web):

1. **Iniciar PostgreSQL**: Asegúrate de que el servidor de base de datos PostgreSQL esté en funcionamiento. En muchos sistemas, PostgreSQL se inicia como servicio automáticamente. Si no, inícialo manualmente (por ejemplo, en Linux `sudo service postgresql start`). Verifica que puedes conectar a la base de datos *urbantracker* (por ejemplo, usando `psql` o PgAdmin) y que las credenciales funcionan.

2. **Levantar el Backend (Spring Boot)**: Inicia el servidor de aplicaciones que provee la API REST y funcionalidades principales. Desde una terminal ubicada en `UrbanTracker/Backend` (si no lo hiciste ya, compila primero con `mvn install`):

3. Ejecuta el comando Maven Spring Boot:

   ```
   mvn spring-boot:run
   ```

   Esto lanzará la aplicación Java. Deberías ver en la consola cómo Spring Boot arranca Tomcat embebido en el puerto 8080 y mensajes de log sobre la inicialización de contexto, conexión a la base de datos, creación de tablas, etc. Si el backend inicia correctamente, el proceso quedará corriendo y escuchando en `http://localhost:8080`. No cierres esta ventana de terminal, déjala abierta para monitorizar logs del servidor.
4. *Tip:* Si prefieres, puedes ejecutar el backend desde un IDE (por ejemplo IntelliJ IDEA o Eclipse) ejecutando la clase `UrbanTrackerApplication`. En cualquier caso, verifica en la consola del backend que aparece la línea *"Started UrbanTrackerApplication in X seconds"* indicando arranque exitoso.
5. Comprueba que el backend responde: abre un navegador o usa *curl* para acceder a `http://localhost:8080/`. Posiblemente veas un error 404 ya que no hay página por defecto (el backend es solo API REST). Sin embargo, puedes acceder a la documentación Swagger-UI que suele estar disponible. Intenta abrir `http://localhost:8080/swagger-ui/index.html` para ver la

documentación interactiva de la API (SpringDoc)[8]. Si se muestra la interfaz de Swagger con el título "UrbanTracker API", el backend está funcionando correctamente.

6. **Levantar el Frontend Web (Next.js)**: Ahora iniciaremos la aplicación web de administración. Abre otra terminal independiente (no detengas el backend) y navega al directorio `UrbanTracker/Web-Admin`. Ejecuta el comando de desarrollo de Next.js:

   ```
   npm run dev
   ```

   Esto iniciará el servidor de desarrollo en modo watch. Next.js usará por defecto el puerto 3000. En la consola verás mensajes indicando que la aplicación está compilando los módulos y luego algo como *"ready - started server on 0.0.0.0:3000"*[9]. Cuando veas **Compiled successfully** la app estará lista.

Ahora abre tu navegador web en **http://localhost:3000**[10]. Deberías ver la página de inicio de UrbanTracker (por ejemplo, un mensaje de *"Bienvenido a UrbanTracker"* con opción de ir al Dashboard)[11]. Desde ahí podrás acceder al panel de administración (es posible que haya una ruta `/Dashboard` para ingresar, probablemente requiera autenticación). Inicia sesión con las credenciales de administrador que creaste. Si aún no hay un usuario admin, crea uno utilizando la API o un método de registro (consulta Solución de errores comunes si tienes problemas en este punto).

1. **(Opcional) Levantar Aplicación Móvil (Expo)**: Si deseas probar la aplicación móvil, puedes ejecutarlas mediante Expo. Este paso requiere que hayas instalado las dependencias móviles y tengas un emulador o dispositivo:
2. *Usando Expo Go en dispositivo físico:* En una terminal, ve al directorio `UrbanTracker/Movil-Driver-Client` y ejecuta:

   ```
   npm start
   ```

   Esto iniciará el servidor Metro Bundler de Expo. La terminal mostrará un código QR. Instala la app **Expo Go** en tu teléfono, ábrela y escanea el QR para cargar la aplicación. Asegúrate que el teléfono esté en la misma red local que tu PC. La app se cargará en modo desarrollo en tu dispositivo. Puedes interactuar con ella, iniciar sesión (si corresponde) o probar sus funciones.
3. *Usando un emulador Android:* Asegúrate de tener un AVD (Android Virtual Device) corriendo. Desde la misma terminal de Expo (Metro Bundler), puedes presionar la tecla **a** para que Expo intente lanzar la app en el emulador Android automáticamente. Alternativamente, ejecuta:

   ```
   npm run android
   ```

Este comando usará la CLI de Expo para compilar e instalar la app en el emulador[12]. La primera vez podría tardar varios minutos ya que compila el código nativo. Si todo va bien, se abrirá la aplicación UrbanTracker en el emulador.

4. Observa la consola de Metro Bundler para cualquier error. Expo recargará la app automáticamente si haces cambios en el código (live reload).
5. ***Nota:*** La app móvil están configuradas para desarrollo, lo que significa que algunas funciones están simuladas. Por ejemplo, puede haber un login de prueba preconfigurado. Verás en la consola logs específicos de la app (p.ej. al hacer login, quizás imprima un token simulado). Esto es normal en modo dev.

6. **Verificación de funcionamiento**: Con backend, frontend y (opcionalmente) apps móviles en marcha, ya tienes UrbanTracker corriendo localmente. Prueba las funciones básicas:

7. En el panel web, intenta crear entidades (rutas, usuarios, vehículos) y asegúrate de que se guarden (puedes comprobar la base de datos para ver si las tablas se llenan).
8. Si configuraste un usuario admin, inicia sesión y navega por el dashboard.
9. En la app móvil, comprueba que puedes iniciar sesión (si el login está implementado) o que al menos la interfaz aparece.
10. Monitorea la consola del backend para verificar que recibe solicitudes cuando interactúas con el frontend o las apps. Cualquier error de servidor aparecerá allí.

Si todos los componentes funcionan como esperado, habrás desplegado exitosamente UrbanTracker en tu entorno local.

7. Solución de errores comunes

A continuación, se listan problemas frecuentes que pueden surgir durante la instalación/ejecución, junto con posibles soluciones:

- **Error: `mvn: command not found` (no se reconoce el comando Maven)** – Este error indica que Maven no está instalado o no está en la variable PATH. Asegúrate de haber instalado Apache Maven correctamente[2]. En Windows, verifica que *MAVEN_HOME* y *Maven\bin* estén en PATH. Como alternativa temporal, usa el wrapper (`mvnw`/`mvnw.cmd`) incluido en el proyecto para ejecutar Maven sin instalación global.

- **Error: `java: command not found` o versión de Java incompatible** – Verifica que tienes Java JDK 17+ instalado y que la variable de entorno *JAVA_HOME*

apunta a la instalación correcta[1]. Si `java -version` muestra una versión menor a 17, instala la versión requerida. En Windows, agrega el binario de Java al PATH. En Mac/Linux, podría ser necesario actualizar alternativamente usando `update-alternatives` (Linux) o configurar en .bash_profile (Mac).

- **Problema: No puedo conectar a la base de datos PostgreSQL** – Si la aplicación arroja errores de conexión (por ejemplo, *"Connection refused"* o *"authentication failed"*):

- Asegúrate de que PostgreSQL esté **en ejecución** y escuchando en el puerto 5432 (u otro puerto si lo cambiaste).
- Verifica las credenciales en `application.properties` o variables de entorno. El error *"password authentication failed for user..."* indica usuario/contraseña incorrectos; corrígelos y reinicia el backend.
- Si PostgreSQL está en otro host (no local), asegúrate de que la URL de conexión apunte a la IP/host correctos y de que el servidor PostgreSQL permita conexiones remotas (configura `postgresql.conf` y `pg_hba.conf` en caso de acceso remoto).
- En algunos sistemas (ej. Ubuntu), la autenticación por default para usuario `postgres` es *peer authentication* (sin contraseña). Podrías crear un usuario con contraseña o cambiar el método de auth a md5. Refiérete a la documentación de PostgreSQL si es el caso.

- **Error al iniciar el backend: *"Timed out waiting for datasource"* o *"Relation does not exist"* –** Esto puede ocurrir si el backend se inicia antes que la base de datos, o si las tablas no se crearon. Asegúrate de iniciar PostgreSQL primero. Si las tablas `users`, `vehicles`, etc., no existen, verifica que `spring.jpa.hibernate.ddl-auto=update` esté habilitado para que se generen automáticamente. Revisa los logs: si ves errores de SQL, quizás la configuración de dialecto o credenciales es incorrecta.

- **El frontend (Next.js) muestra error de** fetch **o CORS** – Si al interactuar con la app web (por ejemplo, al intentar iniciar sesión o cargar datos) ves errores en la consola del navegador sobre *failed to fetch* o CORS, puede deberse a que:

- El frontend no está apuntando al endpoint correcto del backend. Solución: verifica la configuración de la URL de la API (sección de Configuraciones adicionales) y corrígela para que las peticiones vayan a `localhost:8080` (o el host/puerto donde corre el backend).
- El backend está rechazando la petición por CORS. Solución: durante desarrollo, puedes habilitar CORS globalmente en Spring Boot. Por ejemplo, añade un bean de WebMvcConfigurer que permita `http://localhost:3000`. También puedes anotar los controladores/ métodos con `@CrossOrigin(origins = "http://localhost:3000")` para pruebas.

Recuerda que en producción se debe configurar CORS con los dominios adecuados.

- **Error: `npm: command not found`** – Indica que Node.js/NPM no están correctamente instalados en el sistema. Instala Node.js desde el sitio oficial y reinicia la terminal. Comprueba con `node -v` y `npm -v` que estén disponibles[2].

- **Error al ejecutar `npm install` (frontend o móvil)** – A veces NPM puede fallar al resolver dependencias (por ejemplo, por una versión de Node incompatible). Si ocurre, intenta:

- Borrar la carpeta `node_modules` y el archivo `package-lock.json`, luego correr `npm install` de nuevo.
- Asegurarte de usar una versión de Node recomendada (por ejemplo, la LTS). Si el proyecto fue creado con Node 18 y estás usando Node 14, podría haber incompatibilidades.
- Si el error es de permisos (en Linux/macOS, EACCES), evita usar `sudo` con npm. En su lugar, corrige los permisos de tu directorio npm global o utiliza nvm (Node Version Manager) para instalar Node en tu home.

- **Error: *"There is no valid location provider available."*** – Este mensaje puede aparecer en la aplicación móvil al intentar obtener la ubicación GPS[13]. Significa que no se encuentra un proveedor de ubicación válido:

- En un **emulador Android**, probablemente el GPS está desactivado o no hay una ubicación simulada. Solución: habilita la ubicación en el emulador. Ve a la configuración del emulador (Extended Controls > Location) y establece una ubicación virtual. Asegúrate de que en la barra de notificaciones del emulador el ícono de GPS no esté en rojo. También puedes enviar coordenadas manualmente (por ejemplo con Android Studio o usando comandos `adb emu geo fix ...`).
- En un **dispositivo físico**, habilita el GPS: ve a Ajustes > Ubicación y activa la geolocalización (modo alta precisión).
- Este error se detalla en la guía de solución de problemas del proyecto, indicando que simplemente no hay proveedor de GPS activo[13]. Tras activar la ubicación, reinicia la app y debería poder obtener coordenadas.

- **Problemas con permisos de ubicación en Android** – Asegúrate de haber concedido permisos de GPS a la app móvil. En modo Expo, la app solicitará permisos al iniciarse. Si los denegaste accidentalmente, ve a Ajustes de la app en Android y concédele permisos de ubicación. Para desarrollo, se añadieron permisos en el `AndroidManifest.xml` de la app nativa[14], pero igualmente debes aprobarlos en tiempo de ejecución.

- **Error de compilación de la app móvil (Android)** – Si al ejecutar `npm run android` la compilación falla (por ejemplo, por problemas con gradle, dependencias nativas, etc.), puedes intentar limpiar y reconstruir el proyecto Expo:

  ```
  npx expo prebuild --clean
  npx expo run:android
  ```

  Estos comandos forzarán una regeneración de los proyectos nativos Android/iOS a partir de la config Expo[15]. A veces resuelve inconsistencias. Asegúrate también de tener la versión de Java JDK adecuada para compilar el proyecto Android (Expo SDK 53 suele usar JDK 11 internamente; tener JDK 17 no debería ser problema, pero si surgen issues, instalar JDK 11 adicionalmente y apuntar `JAVA_HOME` a JDK 11 durante la compilación nativa puede ayudar).

- **Puertos en uso** – Si al iniciar el backend u otro servicio obtienes errores de "Address already in use", significa que el puerto requerido está ocupado por otro proceso:

- Para el backend (8080): busca qué proceso usa 8080 (`sudo lsof -i :8080` en Linux/Mac, `netstat -a -n -o | find "8080"` en Windows) y detén ese proceso o configura UrbanTracker para usar otro puerto (ver Configuraciones adicionales).
- Para el frontend (3000): quizás tengas otro server Node corriendo. Cierra sesiones anteriores de `npm run dev` o ajusta el puerto.
- Para el Metro bundler de Expo: suele usar 19000/19001. Múltiples instancias de Expo escogerán puertos incrementales automáticamente, pero si por alguna razón no lo hacen, cierra instancias viejas o reinicia el servidor Expo.

- **No puedo iniciar sesión / Acceso denegado** – Si tras levantar todo, no puedes acceder al panel porque no tienes credenciales:

- Revisa si creaste el usuario administrador en la base de datos. Si no, crea uno manualmente (ver sección de datasets necesarios).
- Asegúrate de cifrar la contraseña en BCrypt si insertas el usuario directamente en BD. El backend compara contra una contraseña encriptada. Puedes generar un hash bcrypt usando herramientas en línea o agregando temporalmente código para codificar.
- También verifica qué roles existen. Si el usuario debe tener un rol admin, inserta el rol correspondiente en la tabla `roles` y referencia ese ID en el usuario. El backend puede esperar roles como "ADMIN", "USER", etc. (Revisa la entidad Role en el código para saber los valores posibles).
- Si el sistema tiene un endpoint público de registro, quizás puedas usarlo para crear tu cuenta. Por ejemplo, un POST a `/api/v1/public/user` con los datos

podría crearte un usuario[16]. Asegúrate luego de darle privilegios de admin manualmente si es necesario.

- **Mapas o tiempo real no funcionan** – Si la aplicación carga pero, por ejemplo, el mapa no muestra la ubicación del bus o el seguimiento en tiempo real falla:

- Verifica la configuración de **MQTT/Socket**: UrbanTracker menciona uso de Socket.io/MQTT[17]. Asegúrate de que el broker MQTT (e.g., Mosquitto) esté configurado si corresponde. En requisitos mencionaron Mosquitto, aunque no detallamos su instalación porque podría no ser imprescindible para una demo local. Si el tiempo real no funciona, podría ser porque no se instaló/configuró un broker MQTT. Considera instalar **Mosquitto** y configurar las credenciales/topic que el proyecto use, o si el backend implementa WebSockets, revisar que esté habilitado.
- Verifica las **claves API de mapas**: como se mencionó, si se excedieron las cuotas de la clave incluida, es posible que las peticiones a la API de mapas fallen. Obtén tu propia clave y reemplázala.
- Comprueba la **conectividad del dispositivo**: Si pruebas en un dispositivo físico, este debe poder alcanzar al servidor backend (misma red WIFI). Si no están en la misma red, el dispositivo no encontrará `localhost`. Usa la IP de tu PC en la URL del backend y asegúrate de que firewall/antivirus no bloqueen la conexión.

Finalmente, ante cualquier otro error no listado, **lee atentamente los mensajes en la consola** tanto del backend como de las aplicaciones. Suelen dar pistas de qué está fallando (una dependencia no encontrada, un módulo faltante, etc.). También puedes consultar la documentación oficial de cada tecnología (Spring Boot, Next.js, Expo) para problemas genéricos.

---

[1] [4] pom.xml

https://github.com/AFSB114/UrbanTracker/blob/4e264ccb48444b0dcc8ba8ff80ee6c093d03bb2d/Backend/pom.xml

[2] [3] [17] README.md

https://github.com/AFSB114/UrbanTracker/blob/4e264ccb48444b0dcc8ba8ff80ee6c093d03bb2d/README.md

[9] [10] README.md

https://github.com/AFSB114/UrbanTracker/blob/4e264ccb48444b0dcc8ba8ff80ee6c093d03bb2d/Web-Admin/README.md

[12] package.json

https://github.com/AFSB114/UrbanTracker/blob/4e264ccb48444b0dcc8ba8ff80ee6c093d03bb2d/Movil-User-Client/package.json