

Juan Diego Figari

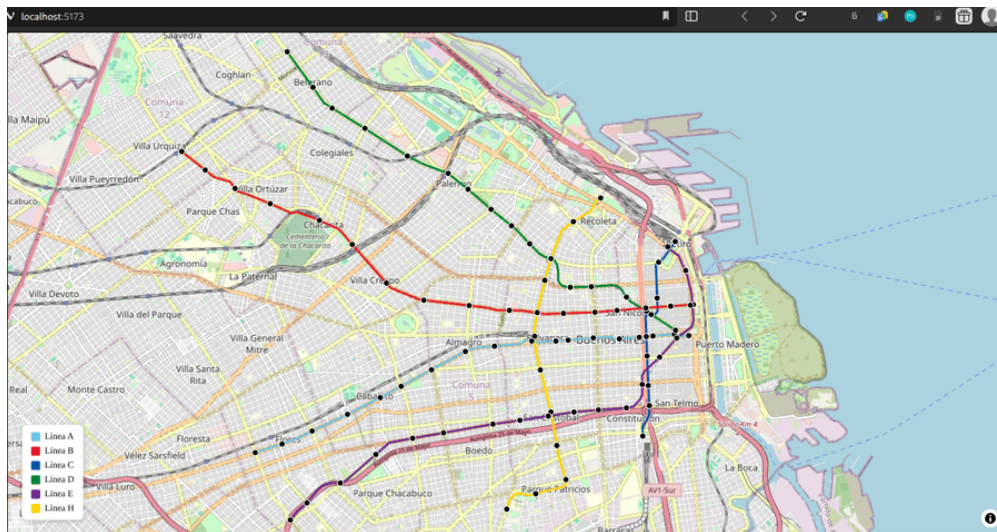
Buenos Aires, Argentina

juancito.diego@gmail.com | <https://github.com/Yarthax23>

Portfolio – Selected Projects

Geospatial Data Visualization — Buenos Aires Subway (Urbanly Exercise) 2026

- Web application rendering subway lines and stations of Buenos Aires using real public datasets from the GCBA.
- Offline data transformation pipeline converting heterogeneous CSV and GeoJSON sources into normalized GeoJSON.
- Deliberate decision to avoid topological reconstruction of line geometries due to missing ordering and connectivity guarantees in the source data.
- Segment-based rendering approach: each line segment is preserved as an independent feature to avoid false connections.
- Focus on correctness, transparency of assumptions, and faithful representation of imperfect real-world data.
- **Repository:** [urbanly-subte-map](#)



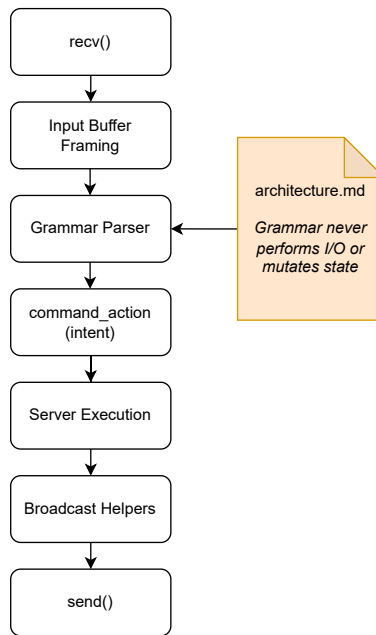
Subway map overview. Lines and stations rendered using MapLibre GL JS from normalized GeoJSON sources. Each segment corresponds directly to original dataset geometry.

- **Key design focus:** Data pipeline separation, explicit assumptions, and correctness over visual overfitting.

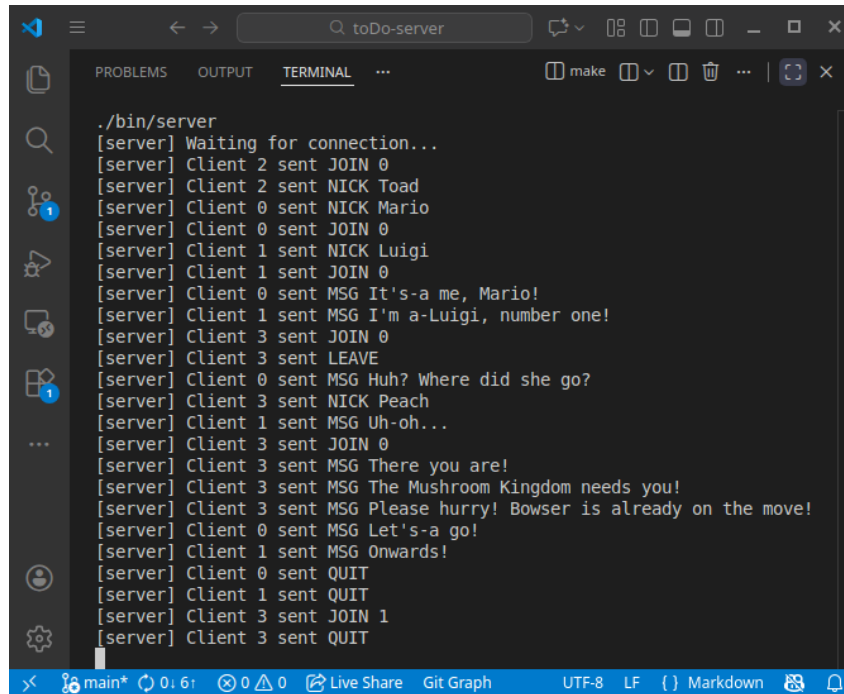
Event-Driven Chat Server in C

2025

- Single-process, event-driven chat server using UNIX domain stream sockets and `select()`.
- Explicit protocol design with strict layered separation: input framing, command grammar, and execution.
- Manual client lifecycle management via a fixed-size client table (single-threaded, no `fork()`).
- Emphasis on correctness, reproducibility, and explicit ownership of server state over premature concurrency.
- **Repository:** [unix-chat-server](#)

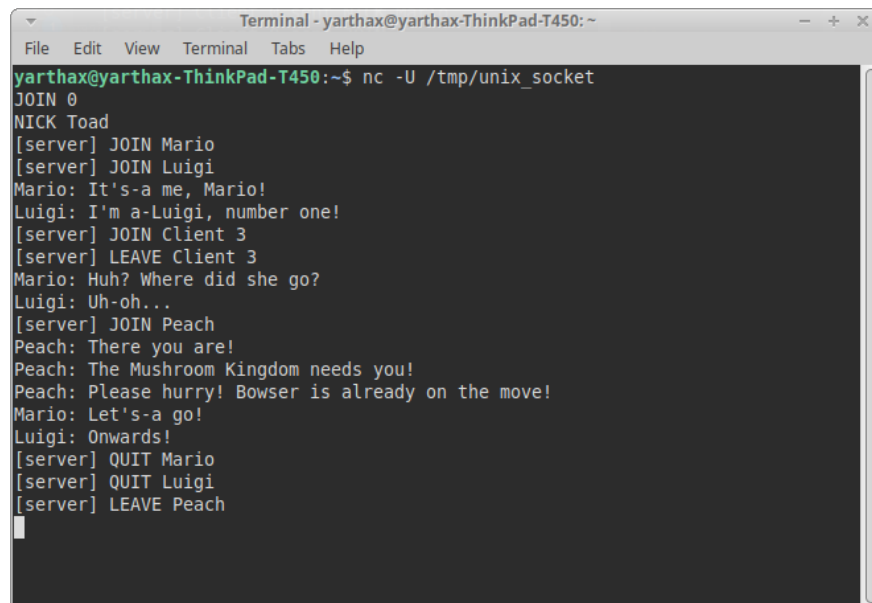


Data flow and separation of responsibilities. Strict pipeline from `recv()` to `send()`, with clear separation between framing, grammar, and execution. The grammar validates input and never performs I/O or state mutation.



```
./bin/server
[server] Waiting for connection...
[server] Client 2 sent JOIN 0
[server] Client 2 sent NICK Toad
[server] Client 0 sent NICK Mario
[server] Client 0 sent JOIN 0
[server] Client 1 sent NICK Luigi
[server] Client 1 sent JOIN 0
[server] Client 0 sent MSG It's-a me, Mario!
[server] Client 1 sent MSG I'm a-Luigi, number one!
[server] Client 3 sent JOIN 0
[server] Client 3 sent LEAVE
[server] Client 0 sent MSG Huh? Where did she go?
[server] Client 3 sent NICK Peach
[server] Client 1 sent MSG Uh-oh...
[server] Client 3 sent JOIN 0
[server] Client 3 sent MSG There you are!
[server] Client 3 sent MSG The Mushroom Kingdom needs you!
[server] Client 3 sent MSG Please hurry! Bowser is already on the move!
[server] Client 0 sent MSG Let's-a go!
[server] Client 1 sent MSG Onwards!
[server] Client 0 sent QUIT
[server] Client 1 sent QUIT
[server] Client 3 sent JOIN 1
[server] Client 3 sent QUIT
```

Server – protocol execution. Event-driven `select()` loop processing a byte stream, emitting JOIN/LEAVE/QUIT events and broadcasting messages to clients in the corresponding room.



```
Terminal - yarthax@yarthax-ThinkPad-T450: ~
File Edit View Terminal Tabs Help
yarthax@yarthax-ThinkPad-T450:~$ nc -U /tmp/unix_socket
JOIN 0
NICK Toad
[server] JOIN Mario
[server] JOIN Luigi
Mario: It's-a me, Mario!
Luigi: I'm a-Luigi, number one!
[server] JOIN Client 3
[server] LEAVE Client 3
Mario: Huh? Where did she go?
Luigi: Uh-oh...
[server] JOIN Peach
Peach: There you are!
Peach: The Mushroom Kingdom needs you!
Peach: Please hurry! Bowser is already on the move!
Mario: Let's-a go!
Luigi: Onwards!
[server] QUIT Mario
[server] QUIT Luigi
[server] LEAVE Peach
```

Spectator client (Toad). Protocol view from a passive client, observing server events and user messages, including a client leaving and rejoining with late nickname assignment.

- **Key design focus:** Ownership of server state, broadcast semantics, and grammar vs execution separation.

Go To-Do Server with PostgreSQL

2025

- REST-style API server written in Go with PostgreSQL persistence.
- Implemented basic CRUD endpoints and database interactions.
- Early Go learning project focused on understanding language fundamentals, standard library usage, and backend structure.
- Project predates adoption of structured commit conventions and formal testing.
- **Repository:** [go-todo-api](#)

Functional Programming Project in Haskell

2025

- University project focused on functional programming using Haskell.
- Implemented core algorithms and data transformations using pure functions and strong type guarantees.
- Emphasis on correctness, immutability, and explicit reasoning about program behavior.
- Early exposure to structured commits and disciplined development practices.
- **Repository:** [Haskell TP \(PLP\)](#)

How I Approach Systems Problems

I prioritize correctness and explicit ownership of state before performance or scalability. I prefer designs where control flow and side effects are visible, debuggable, and easy to reason about. When building systems, I focus on clear boundaries, deterministic behavior, and incremental complexity, delaying concurrency and optimization until semantics are fully understood.

Juan Diego Figari

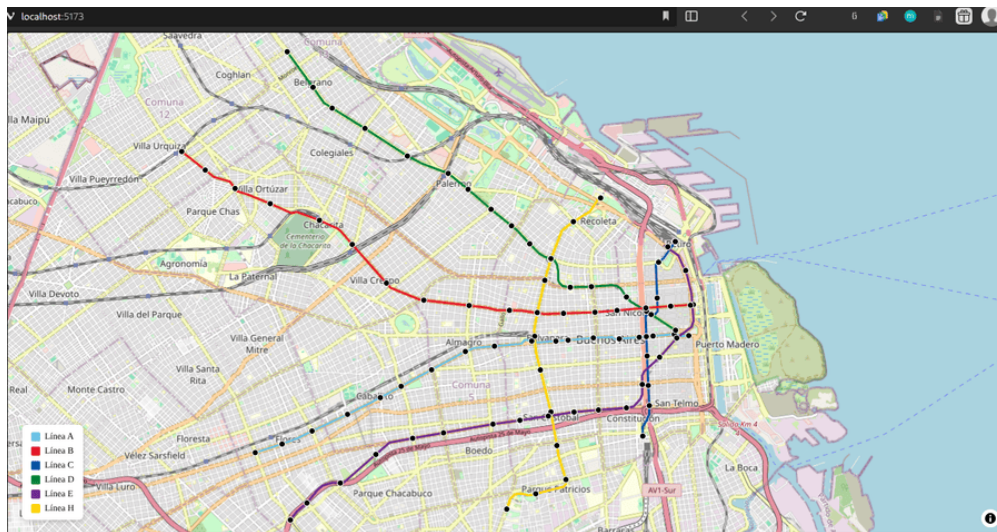
Buenos Aires, Argentina

juancito.diego@gmail.com | <https://github.com/Yarthax23>

Portfolio – Proyectos Seleccionados

Visualización Geoespacial — Subte de Buenos Aires (Ejercicio Urbanly) 2026

- Aplicación web que renderiza las líneas y estaciones del subte de Buenos Aires utilizando datasets públicos reales del GCBA.
- Pipeline offline de transformación de datos para normalizar fuentes heterogéneas en CSV y GeoJSON.
- Decisión explícita de no reconstruir topología debido a la falta de garantías de orden y conectividad en los datos de origen.
- Enfoque de renderizado por segmentos: cada segmento de línea se preserva como un feature independiente para evitar conexiones falsas.
- Énfasis en corrección, transparencia de supuestos y representación fiel de datos reales imperfectos.
- **Repositorio:** [urbanly-subte-map](#)



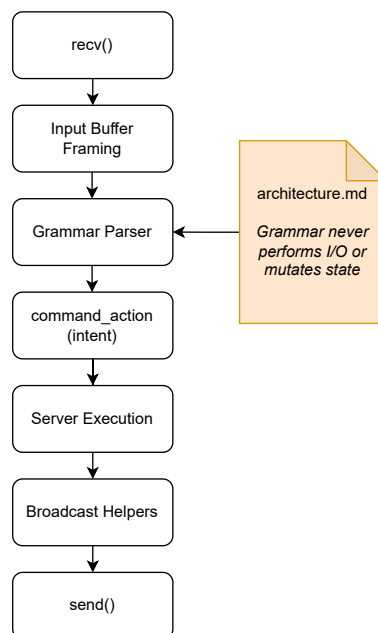
Vista general del mapa. Líneas y estaciones renderizadas con MapLibre GL JS a partir de fuentes GeoJSON normalizadas. Cada segmento corresponde directamente a la geometría original del dataset.

- **Ejes de diseño:** separación del pipeline de datos, supuestos explícitos y corrección por sobre ajuste visual.

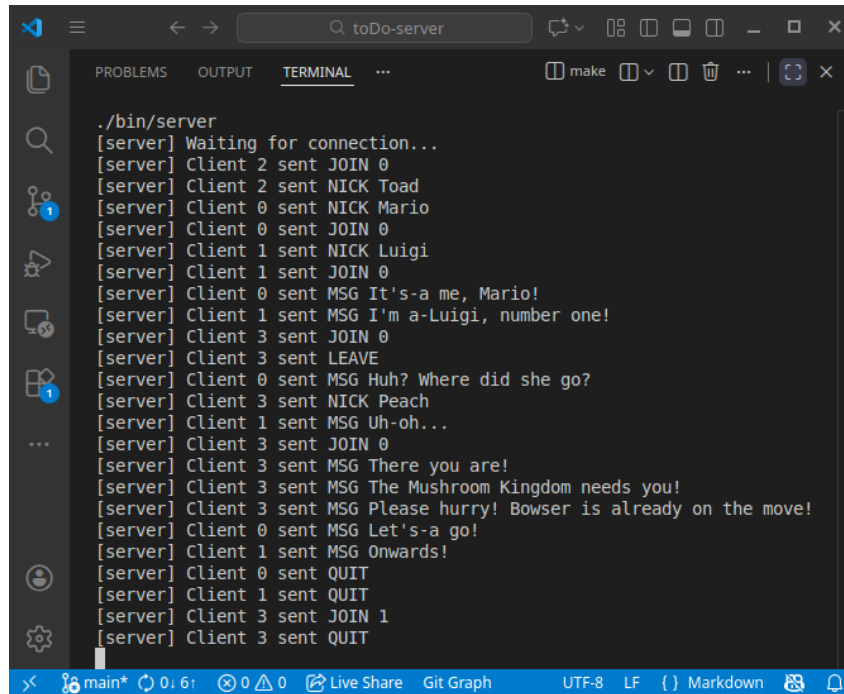
Servidor de Chat Event-Driven en C

2025

- Servidor de chat de proceso único, event-driven, usando sockets UNIX y multiplexación con `select()`.
- Diseño explícito de protocolo con separación estricta por capas: framing de entrada, gramática de comandos y ejecución.
- Gestión manual del ciclo de vida de clientes mediante una tabla de tamaño fijo (single-threaded, sin `fork()`).
- Énfasis en corrección, reproducibilidad y ownership explícito del estado del servidor por sobre concurrencia prematura.
- **Repositorio:** [unix-chat-server](#)

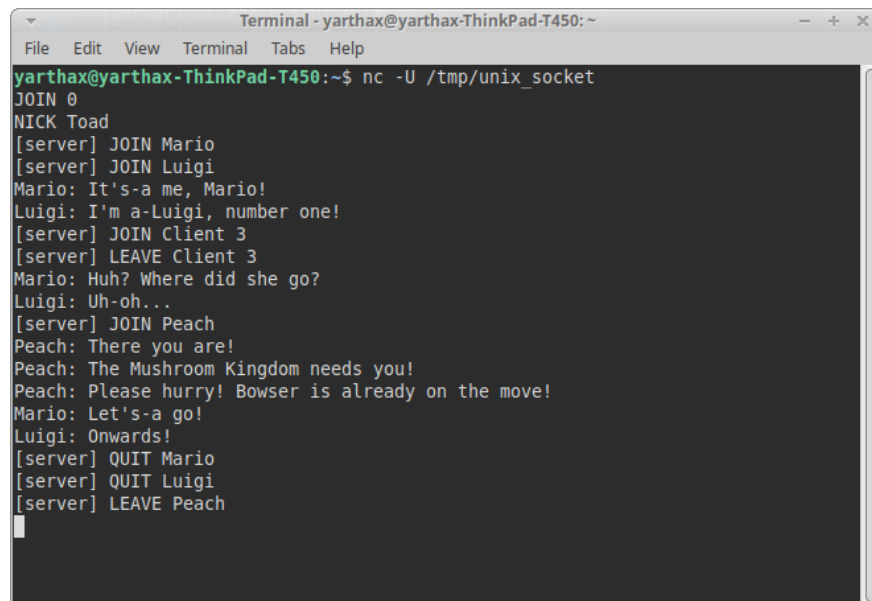


Flujo de datos y separación de responsabilidades. Pipeline estricto desde `recv()` hasta `send()`, con framing, gramática e intención claramente separadas. La gramática valida entradas y nunca realiza I/O ni muta estado.



```
./bin/server
[server] Waiting for connection...
[server] Client 2 sent JOIN 0
[server] Client 2 sent NICK Toad
[server] Client 0 sent NICK Mario
[server] Client 0 sent JOIN 0
[server] Client 1 sent NICK Luigi
[server] Client 1 sent JOIN 0
[server] Client 0 sent MSG It's-a me, Mario!
[server] Client 1 sent MSG I'm a-Luigi, number one!
[server] Client 3 sent JOIN 0
[server] Client 3 sent LEAVE
[server] Client 0 sent MSG Huh? Where did she go?
[server] Client 3 sent NICK Peach
[server] Client 1 sent MSG Uh-oh...
[server] Client 3 sent JOIN 0
[server] Client 3 sent MSG There you are!
[server] Client 3 sent MSG The Mushroom Kingdom needs you!
[server] Client 3 sent MSG Please hurry! Bowser is already on the move!
[server] Client 0 sent MSG Let's-a go!
[server] Client 1 sent MSG Onwards!
[server] Client 0 sent QUIT
[server] Client 1 sent QUIT
[server] Client 3 sent JOIN 1
[server] Client 3 sent QUIT
```

Servidor – ejecución del protocolo. Loop event-driven con `select()`, procesamiento de flujo de bytes, emisión de eventos JOIN/LEAVE/QUIT y broadcast de mensajes a los clientes de una sala.



```
Terminal - yarthax@yarthax-ThinkPad-T450: ~
File Edit View Terminal Tabs Help
yarthax@yarthax-ThinkPad-T450:~$ nc -U /tmp/unix_socket
JOIN 0
NICK Toad
[server] JOIN Mario
[server] JOIN Luigi
Mario: It's-a me, Mario!
Luigi: I'm a-Luigi, number one!
[server] JOIN Client 3
[server] LEAVE Client 3
Mario: Huh? Where did she go?
Luigi: Uh-oh...
[server] JOIN Peach
Peach: There you are!
Peach: The Mushroom Kingdom needs you!
Peach: Please hurry! Bowser is already on the move!
Mario: Let's-a go!
Luigi: Onwards!
[server] QUIT Mario
[server] QUIT Luigi
[server] LEAVE Peach
```

Cliente espectador (Toad). Vista del protocolo desde un cliente pasivo, observando eventos de servidor y mensajes de usuarios, incluyendo salida y reingreso de un cliente con asignación tardía de nickname.

- **Ejes de diseño:** sownership del estado del servidor, semántica de broadcasts y separación entre gramática y ejecución.

Servidor Go con PostgreSQL

2025

- Servidor tipo REST implementado en Go con persistencia en PostgreSQL.
- Implementación de endpoints CRUD básicos e interacción con la base de datos.
- Proyecto inicial de aprendizaje en Go, enfocado en fundamentos del lenguaje y estructura de backends.
- Anterior a la adopción de convenciones de commits estructurados y testing formal.
- **Repositorio:** [go-todo-api](#)

Proyecto de Programación Funcional en Haskell

2025

- Trabajo universitario enfocado en programación funcional utilizando Haskell.
- Implementación de algoritmos y transformaciones de datos mediante funciones puras y garantías fuertes de tipos.
- Enfoque en corrección, inmutabilidad y razonamiento explícito sobre el comportamiento.
- Primer contacto con commits estructurados y prácticas de desarrollo disciplinadas.
- **Repositorio:** [TP Haskell \(PLP\)](#)

Cómo encaro problemas de sistemas

Priorizo la corrección y el ownership explícito del estado antes que el rendimiento o la escalabilidad. Prefiero diseños donde el flujo de control y los efectos laterales sean visibles, depurables y fáciles de razonar. Al construir sistemas, apunto a límites claros, comportamiento determinista y complejidad incremental, postergando concurrencia y optimización hasta que la semántica esté bien definida.