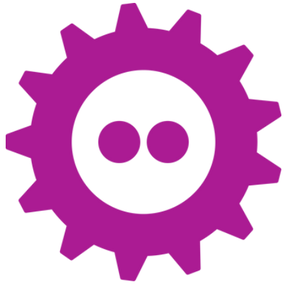
MASTERING **OBSERVABILITY** WITH **SIGNOZ**

Angeles MORA

FOSDEM 2025

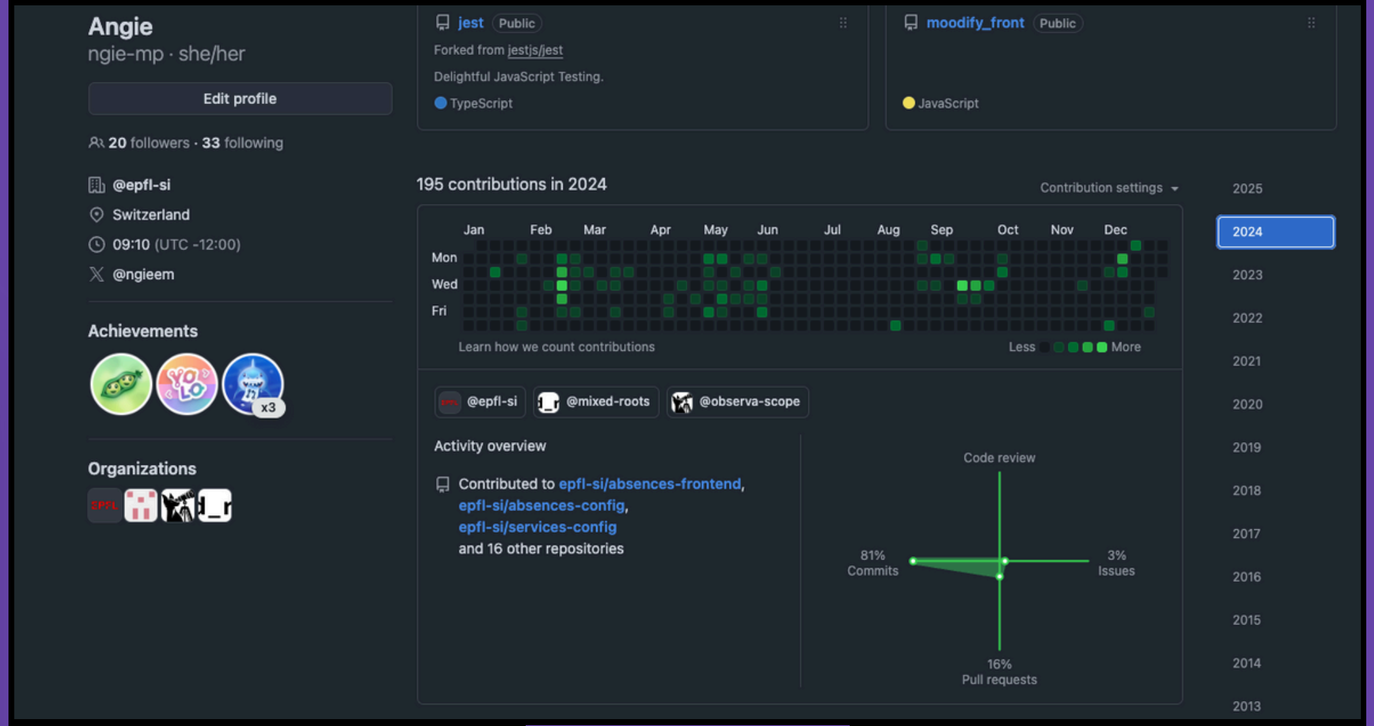
02.022025

WELCOME TO MY FIRST **IT_TALK** EVER

ABOUT ME

```
const Angie: Person = {
  age: 33,
  profession: 'Software Engineer',
  characteristics: [
    'benevolent',
    'single mom and dog-mom',
    'passionate',
    'enthusiastic',
  ],
  activistIn: [
    'teamwork',
    'UX design',
    'communication',
    'innovation',
  ],
  goals: [
    'Lead a Team',
    'Learn AI/ML/LLM & +Observability',
    'Transition to Solution Engineer',
  ],
  facts: [
    'Exploring observability tools for frontend and backend performance.',
    'Searching for adapted solutions to improve system monitoring.',
    'Learning OpenTelemetry for better tracing and logs.'
  ],
};
```



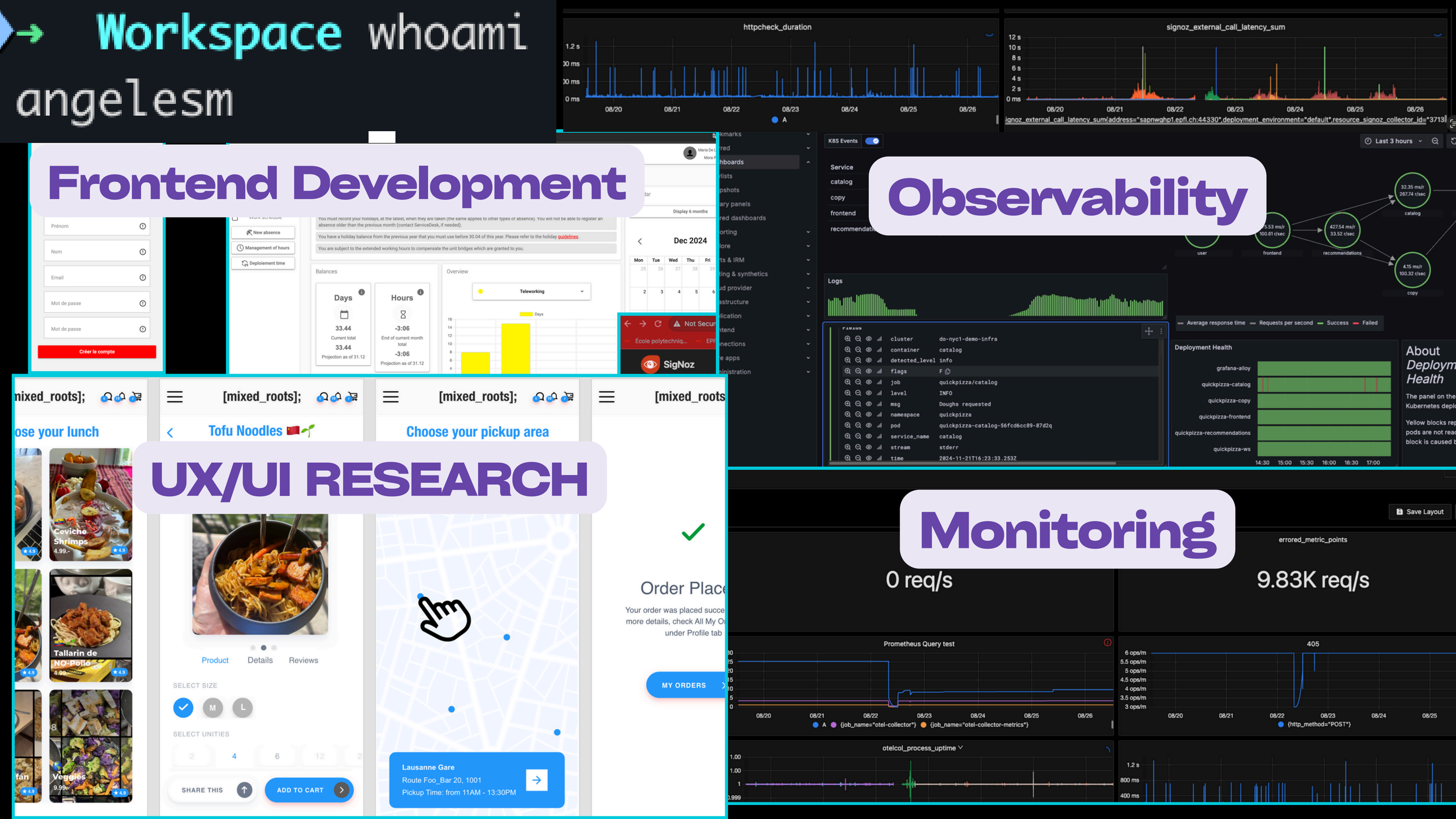
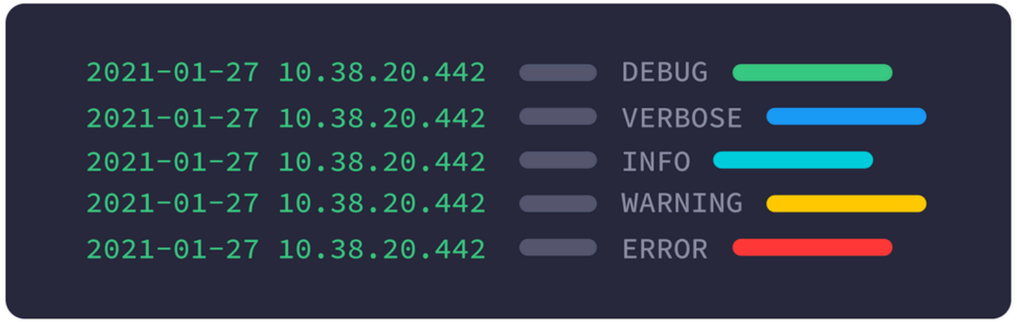

Workspace whoami
angelesm
Frontend Development
Observability
UX/UI RESEARCH
Monitoring
httpcheck_duration
signoz_external_call_latency_sum
Prénom
Nom
Email
Mot de passe
Mot de passe
Créer le compte
New absence
Management of hours
Deploiement time
Balances
Days
Hours
33.44
-3:06
Overview
Teleworking
Dec 2024
SigNoz
K8S Events
Last 3 hours
Service
catalog
copy
frontend
recommendation
Logs
cluster do-nyc1-demo-infra
container catalog
detected_level info
flags F
job quickpizza/catalog
level INFO
msg Doughs requested
namespace quickpizza
pod quickpizza-catalog-56fcd6cc89-87d2q
service_name catalog
stream stderr
time 2024-11-21T16:23:33.253Z
Average response time
Requests per second
Success
Failed
Deployment Health
grafana-alloy
quickpizza-catalog
quickpizza-copy
quickpizza-frontend
quickpizza-recommendations
quickpizza-ws
About Deployment Health
[mixed_roots];
Tofu Noodles
Choose your pickup area
Ceviche Shrimps
Tallarin de NO-Pollo
Veggies
Product
Details
Reviews
SELECT SIZE
SELECT UNITIES
SHARE THIS
ADD TO CART
Lausanne Gare
Route Foo_Bar 20, 1001
Pickup Time: from 11AM - 13:30PM
Order Place
MY ORDERS
Save Layout
errored_metric_points
0 req/s
9.83K req/s
Prometheus Query test
405
otelcol_process_uptime

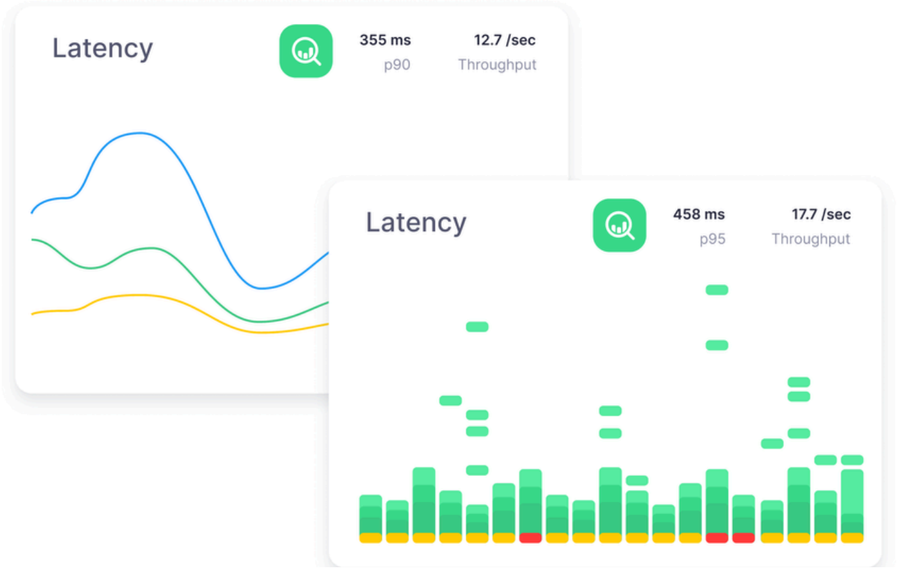
TABLE OF CONTENT

Imagine Observability Like Cooking

Logs



Metrics

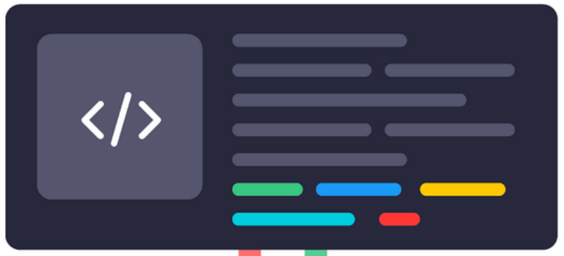


Traces

THE OBSERVABILITY JOBS TO BE DONE

Instrumentation

Generating the data that enables you to ask questions

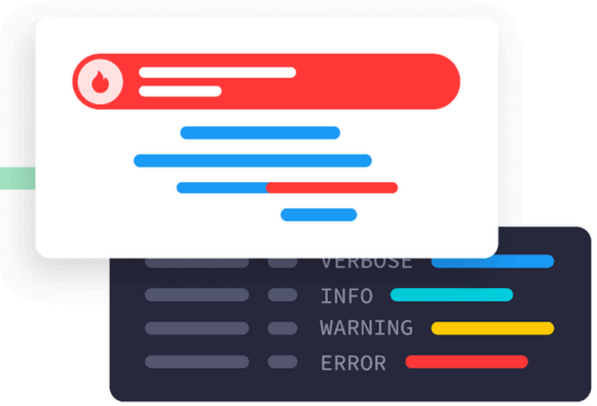


Monitoring

Asking the same question, over and over again

Debugging

Asking new questions on the fly



Identify 🔍

Determine what needs to be observed (key metrics, logs, traces).
Define key performance indicators (KPIs) and service-level objectives (SLOs).
Understand system dependencies and critical paths.

Collect 📊

Gather relevant data from logs, metrics, and distributed traces.
Use telemetry tools like OpenTelemetry, Prometheus.
Ensure data is structured, timestamped, and contextually rich.

Do (Act & Analyze)⚡

Correlate and analyze the collected data to detect anomalies.
Implement alerts and automated responses for faster issue resolution.
Continuously improve the system using insights from observability data.

INTALLING SIGNOZ

What do we need to have Self-Hosted?

- Ensure your system is Linux or macOS.
- For macOS, manually install Docker Engine
- On Linux, the install script will handle it.
- Minimum 4GB of memory allocated to Docker.
- Open ports: 3301, 4317, and 4318.
- Git client installed.

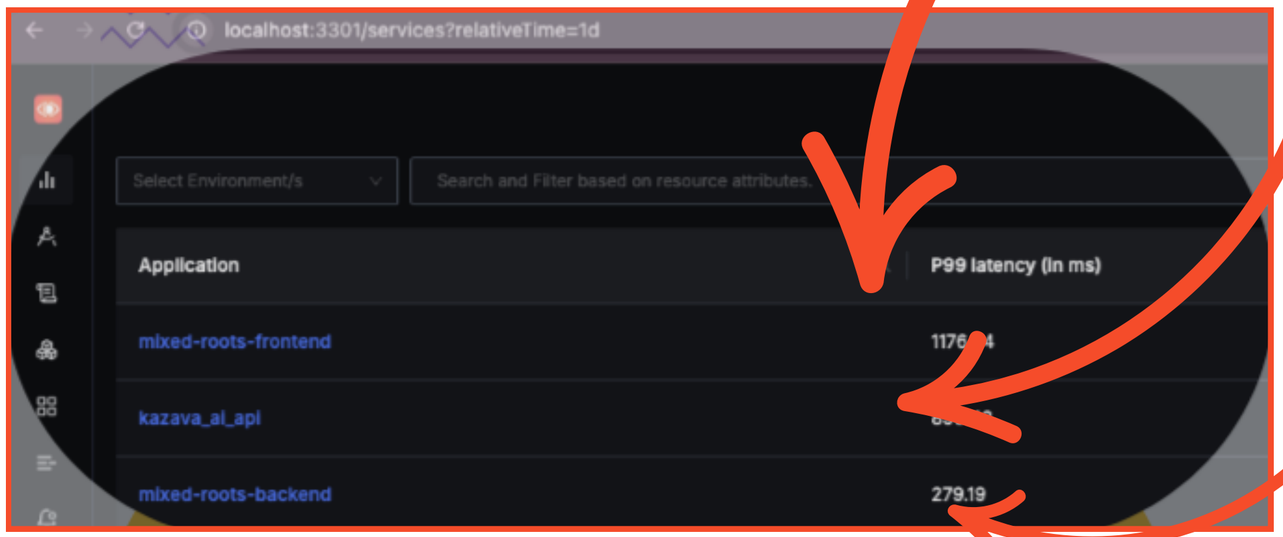

```
git clone -b main https://github.com/SigNoz/signoz.git &&
cd signoz/deploy/ ./install.sh
```

```
git clone -b main https://github.com/SigNoz/signoz.git && cd
signoz/deploy/docker docker compose up -d --remove-orphans
```

CONTAINERS RUNNING AFTER THE INSTALL

```
[+] Running 10/10
 ✔ Network signoz-net                Created
                         0.0s
 ✔ Container signoz-zookeeper-1      Healthy
                        31.0s
 ✔ Container signoz-init-clickhouse  Exited
                         2.0s
 ✔ Container signoz-clickhouse       Healthy
                        62.3s
 ✔ Container schema-migrator-sync    Exited
                        65.7s
 ✔ Container schema-migrator-async   Started
                        65.8s
 ✔ Container signoz-query-service    Healthy
                        96.4s
 ✔ Container signoz-otel-collector   Started
                        96.5s
 ✔ Container signoz-alertmanager     Started
                        96.5s
 ✔ Container signoz-frontend         Started
                        96.6s
➜  docker git:(main) ✗ docker ps
CONTAINER ID   IMAGE                                        COMMAND                  CREATED        STATUS                  PORTS                                                            NAMES
f187390df42e   signoz/frontend:0.70.0                       "nginx -g 'daemon of…"   22 hours ago   Up 22 hours             80/tcp, 0.0.0.0:3301->3301/tcp                                   signoz-frontend
6d7a495caec9   signoz/alertmanager:0.23.7                   "/bin/alertmanager -…"   22 hours ago   Up 22 hours             9093/tcp                                                         signoz-alertmanager
4b5526aed795   signoz/signoz-otel-collector:0.111.24        "/signoz-collector -…"   22 hours ago   Up 22 hours             0.0.0.0:4317-4318->4317-4318/tcp, 0.0.0.0:8082->8082/tcp         signoz-otel-collector
eeec06b85d54   signoz/query-service:0.70.0                  "./query-service --c…"   22 hours ago   Up 22 hours (healthy)   8080/tcp                                                         signoz-query-service
a98e07b38980   clickhouse/clickhouse-server:24.1.2-alpine   "/entrypoint.sh"         22 hours ago   Up 22 hours (healthy)   8123/tcp, 9000/tcp, 9009/tcp                                     signoz-clickhouse
5f4e7c1bc352   bitnami/zookeeper:3.7.1                      "/opt/bitnami/script…"   22 hours ago   Up 22 hours (healthy)   2181/tcp, 2888/tcp, 3888/tcp, 8080/tcp                           signoz-zookeeper-1
d8bb6eb3e500   postgres:15                                  "docker-entrypoint.s…"   2 weeks ago    Up 2 weeks              0.0.0.0:5432->5432/tcp                                           postgres_container_mixed_roots
```

INSTRUMENTATION WITH OPENTELEMETRY



FRONTEND:
NEXT.JS

API:
PYTHON + FLASK

BACKEND:
NEST.JS

INSTRUMENTATION OF FRONTEND -> NEXT.JS

```
"node -r ./otel-setup.js node_modules/.bin/next dev -p 3001"
```

```
},
"dependencies": {
  "@opentelemetry/api": "^1.9.0",
  "@opentelemetry/api-logs": "^0.57.0",
  "@opentelemetry/auto-instrumentations-node":
  "@opentelemetry/exporter-otlp-grpc": "^0.26.
  "@opentelemetry/exporter-trace-otlp-http": "
  "@opentelemetry/instrumentation": "^0.57.0",
  "@opentelemetry/instrumentation-fetch": "^0.
  "@opentelemetry/instrumentation-http": "^0.5
  "@opentelemetry/sdk-logs": "^0.57.0",
  "@opentelemetry/sdk-node": "^0.56.0",
  "@opentelemetry/sdk-trace-base": "^1.30.0",
  "@opentelemetry/sdk-trace-web": "^1.30.0",
  "@opentelemetry/semantic-conventions": "^1.2
  "@tailwindcss/forms": "^0.5.9",
  "@tailwindcss/typography": "^0.5.15",
  "@types/express": "^5.0.0",
  "@vercel/otel": "^1.10.0",
  "axios": "^1.7.9",
  "chart.js": "^4.4.7",
  "express": "^4.21.2",
  "install": "^0.13.0",
  "next": "15.0.3",
  "react": "^18.3.1",
```

```
const { WebTracerProvider } = require('@opentelemetry/sdk-trace-web');  56.1k (gzipped: 15.3k)
const { BatchSpanProcessor } = require('@opentelemetry/sdk-trace-base');  49.3k (gzipped: 12.7k)
const { FetchInstrumentation } = require('@opentelemetry/instrumentation-fetch');  89k (gzipped: 25.5k)
const { registerInstrumentations } = require('@opentelemetry/instrumentation');  77.2k (gzipped: 22k)
const { OTLPTraceExporter } = require('@opentelemetry/exporter-trace-otlp-http');  20.6k (gzipped: 6.3k)
const { Resource } = require('@opentelemetry/resources');  28.9k (gzipped: 6.3k)
const { SemanticResourceAttributes } = require('@opentelemetry/semantic-conventions');  35.2k (gzipped: 1
const { diag, DiagConsoleLogger, DiagLogLevel } = require('@opentelemetry/api');  19.8k (gzipped: 5.5k)

// Set diagnostics for debugging
diag.setLogger(new DiagConsoleLogger(), DiagLogLevel.INFO);

// Dynamic service name
const serviceName = process.env.OTEL_SERVICE_NAME || 'mixed-roots-frontend';
console.log('Using service name:', serviceName);

// Initialize Tracer Provider
const provider = new WebTracerProvider({
o add a breakpoint  esource({
    [SemanticResourceAttributes.SERVICE_NAME]: serviceName,
  }),
});

// OTLP Trace Exporter
const otlpExporter = new OTLPTraceExporter({
  url: 'http://localhost:4318/v1/traces', // Update as needed
});
provider.addSpanProcessor(new BatchSpanProcessor(otlpExporter));

// Fetch Instrumentation (for browser apps)
registerInstrumentations({
  instrumentations: [new FetchInstrumentation()],
```

INSTRUMENTATION OF API -> PYTHON + FLASK

```
(venv) ➜  foodAI git:(main) ✗ OTEL_RESOURCE_ATTRIBUTES=service.name=kazava_ai_api OTEL_EXPORTER_OTLP_ENDPOINT="http://127.0.0.1:4317" OTEL_EXPORTER_OTLP_PROTOCOL=grpc opentelemetry-instrument  flask run --host 0.0.0.0 --port 8084
Overriding of current TracerProvider is not allowed
Attempting to instrument Flask app while already instrumented
Some weights of BertForSequenceClassification were not initialized from the model checkpoint at bert-base-uncased and are newly initialized: ['classifier.bias', 'classifier.weight']
You should probably TRAIN this model on a down-stream task to be able to use it for predictions and inference.
 * Debug mode: off
INFO:werkzeug:WARNING: This is a development server. Do not use it in a production deployment. Use a production WSGI server instead.
 * Running on all addresses (0.0.0.0)
 * Running on http://127.0.0.1:8084
 * Running on http://192.168.49.40:8084
INFO:werkzeug:Press CTRL+C to quit
```

```
➜  foodAI git:(main) ✗ bat requirements.txt
       File: requirements.txt
   1   flask
   2   flask-cors
   3   transformers
   4   torch
   5   opentelemetry-api
   6   opentelemetry-sdk
   7   opentelemetry-instrumentation-flask
```

app.py

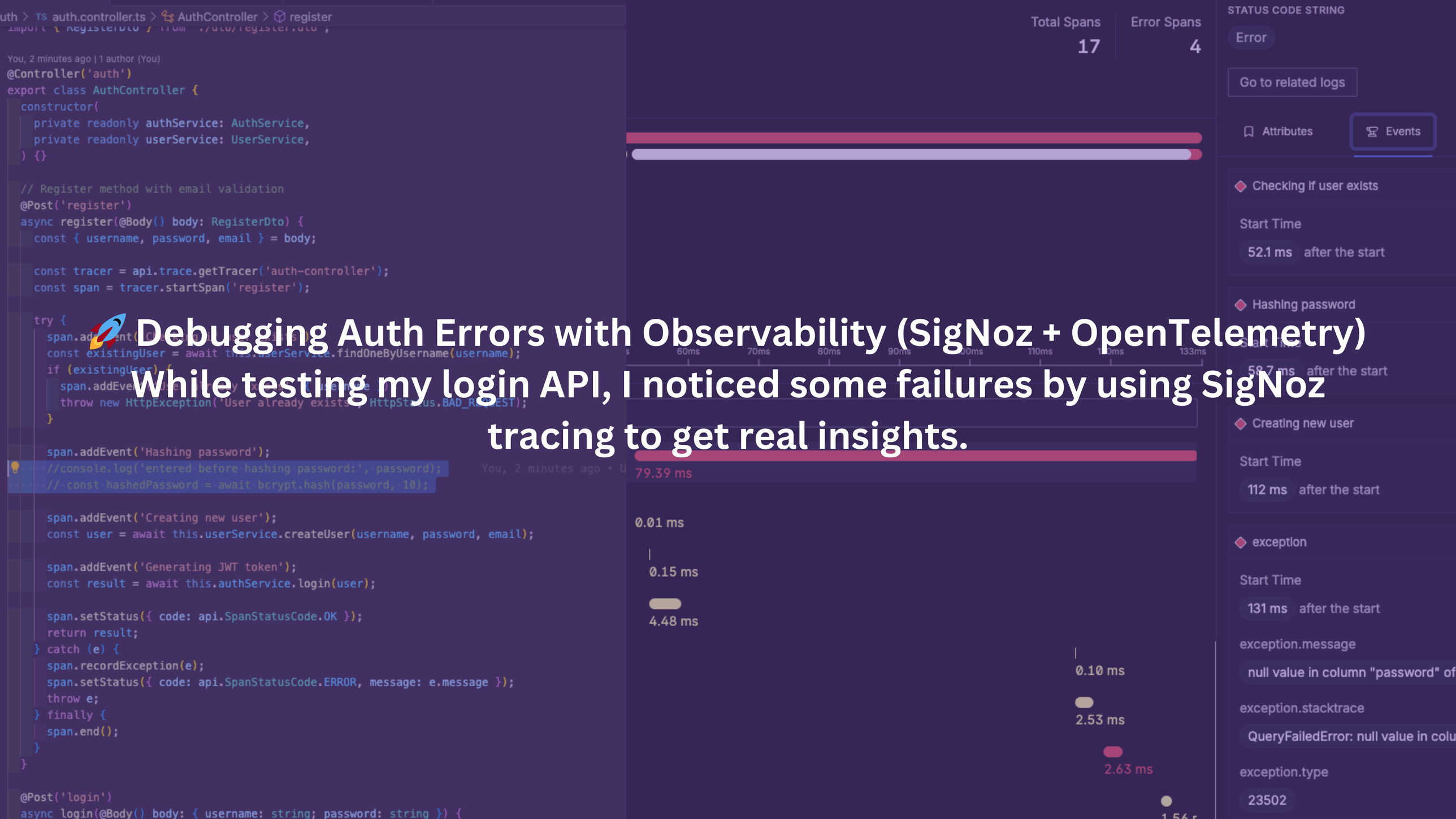
You, 4 weeks ago | 1 author (You)

```python
from flask import Flask, request, jsonify
from flask_cors import CORS
import torch
from transformers import AutoTokenizer, AutoModelForSequenceClassification
import json
import logging
from opentelemetry import trace
from opentelemetry.instrumentation.flask import FlaskInstrumentor
from opentelemetry.sdk.trace import TracerProvider
from opentelemetry.sdk.trace.export import BatchSpanProcessor
from opentelemetry.exporter.otlp.proto.grpc.trace_exporter import OTLPSpanExporter

# Set up OpenTelemetry tracing
trace.set_tracer_provider(TracerProvider())
tracer = trace.get_tracer(__name__)

# Set up OTLP exporter
otlp_exporter = OTLPSpanExporter(endpoint="localhost:4317", insecure=True)
span_processor = BatchSpanProcessor(otlp_exporter)
trace.get_tracer_provider().add_span_processor(span_processor)

# Flask setup
app = Flask(__name__)
FlaskInstrumentor().instrument_app(app)
CORS(app, origins=[
    "http://localhost:3001",
    "http://12.0.0.1:3005",
    "http://127.0.0.1:82"
])
    You, 4 weeks ago • [wip] observability
# Logging configuration
logging.basicConfig(level=logging.INFO)
logger = logging.getLogger(__name__)

# Load Hugging Face model and tokenizer
try:
    with tracer.start_as_current_span("model_loading"):
        model_name = "bert-base-uncased"  # Replace with a fine-tuned model
        tokenizer = AutoTokenizer.from_pretrained(model_name)
        model = AutoModelForSequenceClassification.from_pretrained(model_name)
except Exception as e:
    current_span = trace.get_current_span()
    current_span.add_event("Error loading model", {"error_message": str(e)})
    logger.error(f"Error loading Hugging Face model: {str(e)}")
    raise

# Load the meal data
try:
    with tracer.start_as_current_span("load_meal_data"):
        with open('data.json') as f:
```

INSTRUMENTATION OF BACKEND -> NEST.JS

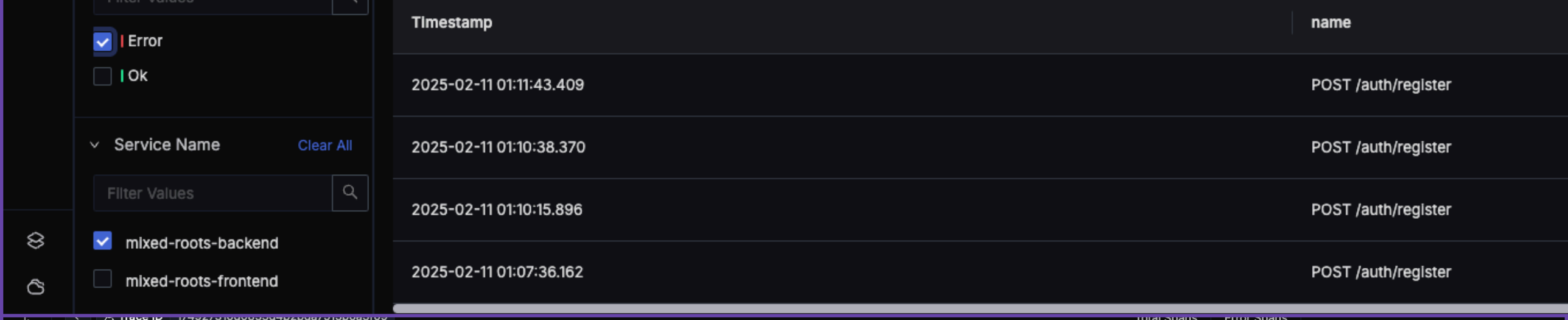
```typescript
import { getNodeAutoInstrumentations } from '@opentelemetry/auto-instrumentations-node';
import { OTLPTraceExporter } from '@opentelemetry/exporter-trace-otlp-http';  12.2k (gzipped: 3.9k)
import { Resource } from '@opentelemetry/resources';  4.1k (gzipped: 1.6k)
import * as opentelemetry from '@opentelemetry/sdk-node';
import { SemanticResourceAttributes } from '@opentelemetry/semantic-conventions';  2.3k (gzipped: 1
import {
  PeriodicExportingMetricReader,
  ConsoleMetricExporter,
} from '@opentelemetry/sdk-metrics';  36k (gzipped: 8.4k)
import { diag, DiagConsoleLogger, DiagLogLevel } from '@opentelemetry/api';  5.5k (gzipped: 2k)
import { HttpInstrumentation } from '@opentelemetry/instrumentation-http';
import { PgInstrumentation } from '@opentelemetry/instrumentation-pg';

const init = function (serviceName: string) {
  diag.setLogger(new DiagConsoleLogger(), DiagLogLevel.INFO);

  // Define traces
  const traceExporter = new OTLPTraceExporter({
    url: 'http://localhost:4318/v1/traces',
  });

  const sdk = new opentelemetry.NodeSDK({
    traceExporter,
    metricReader: new PeriodicExportingMetricReader({
      exporter: new ConsoleMetricExporter(),
    }),
    instrumentations: [
      getNodeAutoInstrumentations(),
      new HttpInstrumentation(),
      new PgInstrumentation(),
    ],
    resource: new Resource({
      [SemanticResourceAttributes.SERVICE_NAME]: serviceName,
    }),      You, 57 seconds ago • Uncommitted changes
  });

  sdk.start();

  process.on('SIGTERM', () => {
    sdk
```

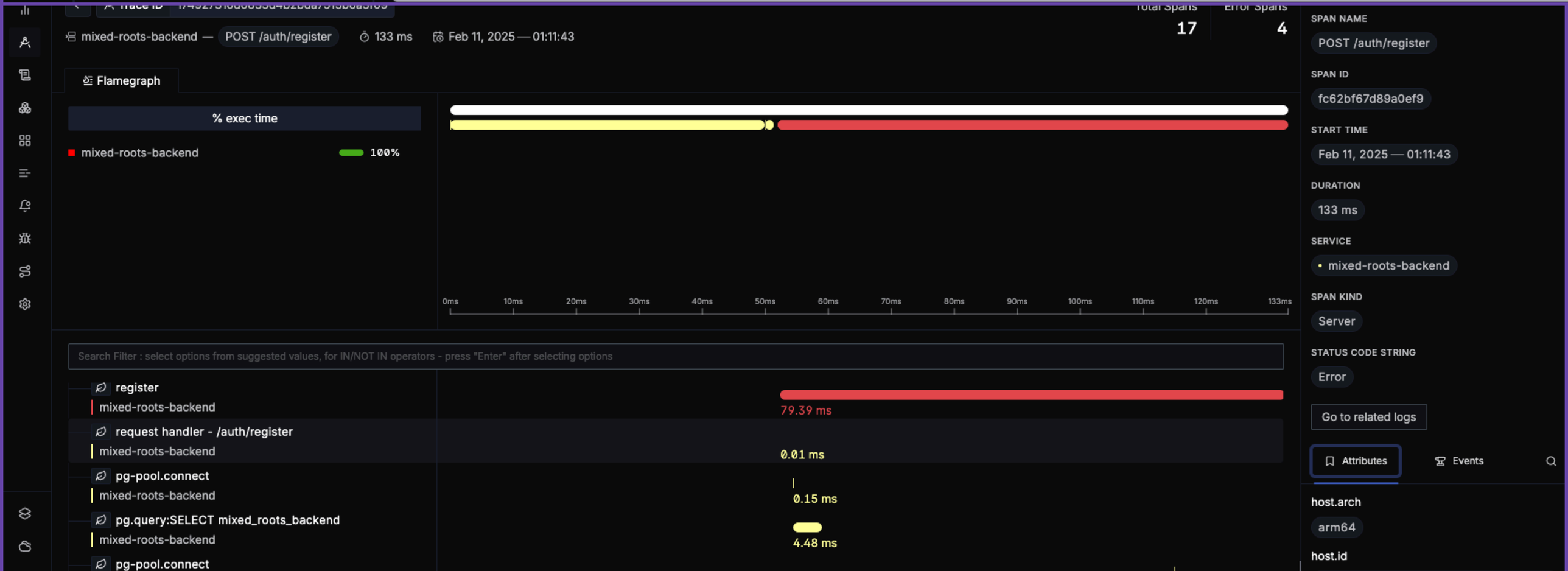
```typescript
You, 39 minutes ago | 1 author (You)
import { NestFactory } from '@nestjs/core';
import { AppModule } from './app.module';
import { WinstonLoggerService } from './logger/winston-logger.service';

import init from './tracers';

// Initialize OpenTelemetry SDK before any NestJS modules are loaded
init('mixed-roots-backend');

async function bootstrap() {
  // Ensure OpenTelemetry is initialized first
  console.log('OpenTelemetry is initialized.');

  // Create NestJS application
  const app = await NestFactory.create(AppModule, { cors: true });
  const customLogger = new WinstonLoggerService();
  customLogger.log('Application is starting...');

  // Enable CORS
  app.enableCors({
    origin: 'http://localhost:3001', // URL of your Next.js app
    credentials: true, // Allow cookies or authentication tokens
    allowedHeaders: ['traceparent', 'tracestate', 'content-type'],
  });

  console.log('NestJS application is about to listen on port 3005...');
  //app.useGlobalPipes(new ValidationPipe({ transform: true }));
      You, 39 minutes ago • Uncommitted changes
  // Start the NestJS app
  await app.listen(3005);
}

bootstrap();
```



🚀 Debugging Auth Errors with Observability (SigNoz + OpenTelemetry)

While testing my login API, I noticed some failures by using SigNoz tracing to get real insights.


Error
Ok
Service Name
Clear All
Filter Values
mixed-roots-backend
mixed-roots-frontend
Timestamp
name
2025-02-11 01:11:43.409
POST /auth/register
2025-02-11 01:10:38.370
POST /auth/register
2025-02-11 01:10:15.896
POST /auth/register
2025-02-11 01:07:36.162
POST /auth/register


mixed-roots-backend — POST /auth/register
133 ms
Feb 11, 2025 — 01:11:43
Total Spans
17
Error Spans
4
Flamegraph
% exec time
mixed-roots-backend
100%
0ms
10ms
20ms
30ms
40ms
50ms
60ms
70ms
80ms
90ms
100ms
110ms
120ms
133ms
Search Filter : select options from suggested values, for IN/NOT IN operators - press "Enter" after selecting options
register
mixed-roots-backend
79.39 ms
request handler - /auth/register
mixed-roots-backend
0.01 ms
pg-pool.connect
mixed-roots-backend
0.15 ms
pg.query:SELECT mixed_roots_backend
mixed-roots-backend
4.48 ms
pg-pool.connect
SPAN NAME
POST /auth/register
SPAN ID
fc62bf67d89a0ef9
START TIME
Feb 11, 2025 — 01:11:43
DURATION
133 ms
SERVICE
mixed-roots-backend
SPAN KIND
Server
STATUS CODE STRING
Error
Go to related logs
Attributes
Events
host.arch
arm64
host.id

CONCLUSION

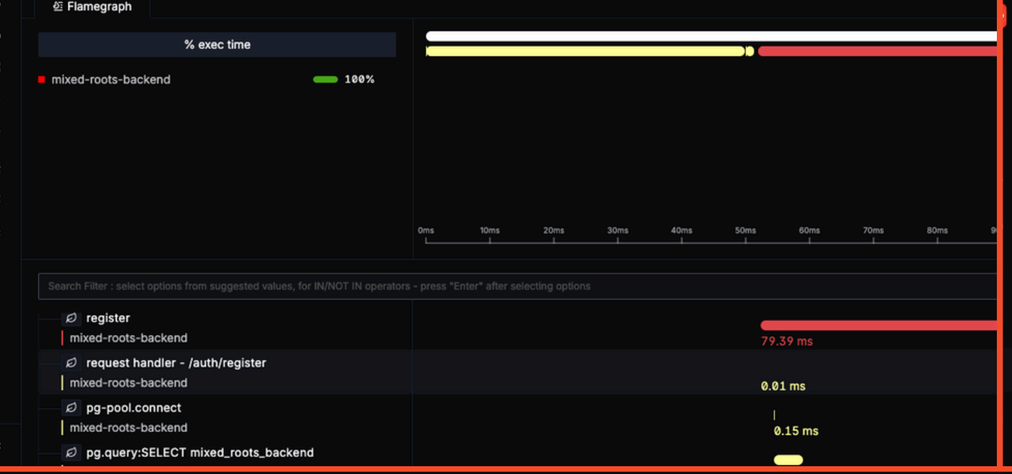
Debugging Without Observability is Like Cooking Blindfolded >.<

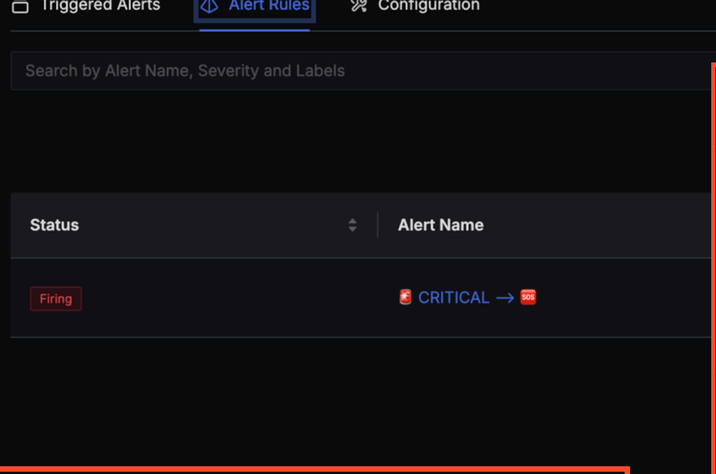
<NG>
Imagine trying to cook a perfect dish without seeing, smelling, or tasting, just hoping it turns out right.
That's what debugging without observability feels like!
</NG>

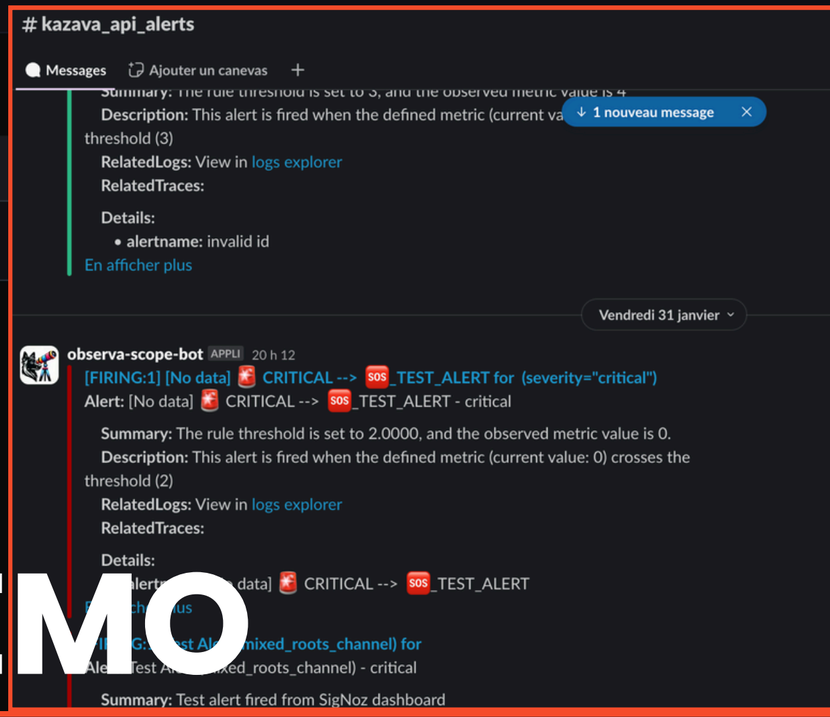
I instrumented my backend with SigNoz + OpenTelemetry, and boom 💥 instead of blindly guessing why logins failed, I got 👩‍💻 :

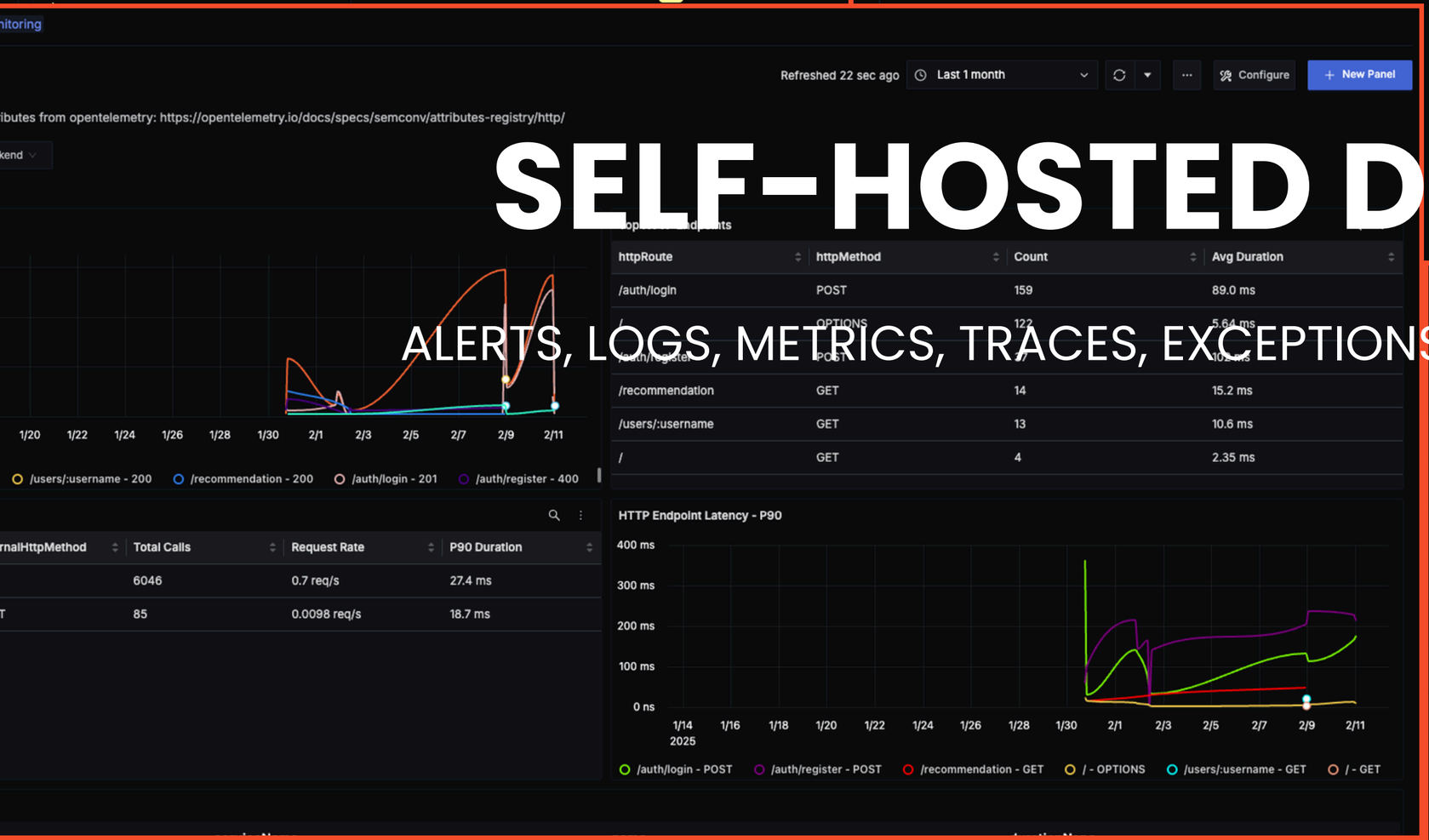
- Precise error tracking (invalid credentials spotted instantly)
- Step-by-step execution flow (no more black-box debugging)
- Real-time insights to fix issues fast

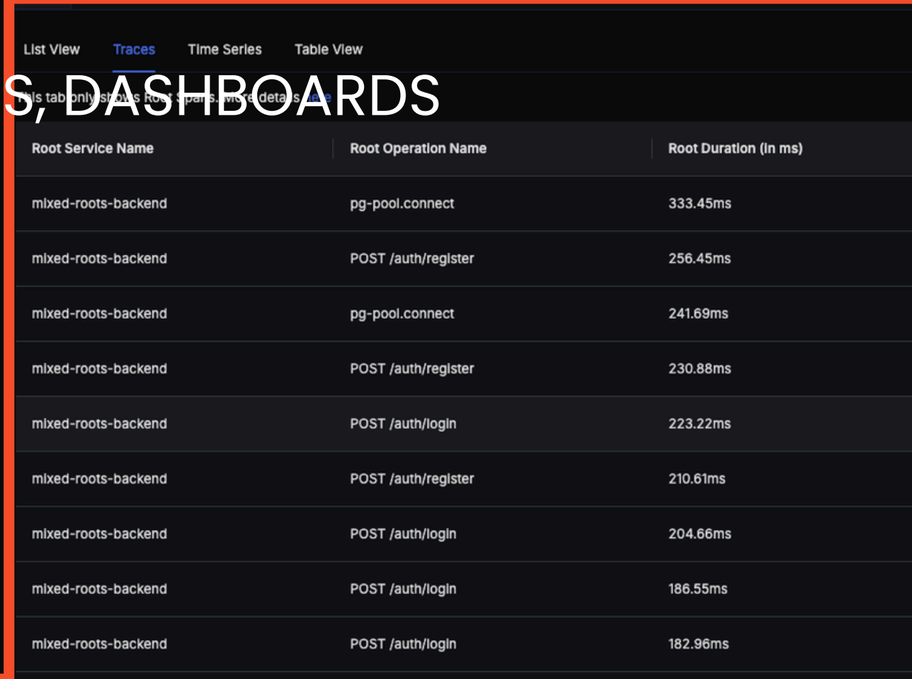
🔍 Observability isn't a luxury, it's the recipe for reliable systems. No more guesswork, just data-driven fixes!











SELF-HOSTED DEMO

ALERTS, LOGS, METRICS, TRACES, EXCEPTIONS, DASHBOARDS

SIGNOZ DASHBOARDS

Available Dashboards

Below is a list of available dashboard templates in this repository:

- Hostmetrics Dashboard: Monitors general host metrics, including CPU, memory, and disk usage.
- Kubernetes Infra Dashboard: Visualizes metrics related to Kubernetes infrastructure.
- Key Operations Dashboard: Tracks key operations within an application, focusing on performance and reliability.
- Apache Web Server Dashboard: Monitors Apache web server metrics like request rates and active connections.
- APM Dashboard: Visualizes application performance metrics, including latency, throughput, and error rates.
- Docker Container Metrics Dashboard: Tracks metrics related to Docker containers, such as CPU and memory usage.
- CouchDB Dashboard: Monitors CouchDB-specific metrics, such as document read/write rates.
- ECS Infrastructure Metrics Dashboard: Visualizes metrics for Amazon ECS infrastructure.
- Flask Monitoring Dashboard: Monitors performance metrics for Flask applications.
- HAProxy Dashboard: Monitors HAProxy metrics such as request rates and active sessions.
- Jenkins Dashboard: Tracks Jenkins metrics, including job success rates and queue times.
- JMX Dashboard: Monitors Java Management Extensions (JMX) metrics.
- JVM Dashboard: Tracks JVM metrics, including heap usage, garbage collection, and thread counts.
- Memcached Dashboard: Visualizes Memcached-specific metrics, such as cache hit/miss rates.
- MongoDB Dashboard: Monitors MongoDB operations, memory usage, and performance metrics.
- MySQL Dashboard: Tracks MySQL metrics, including queries per second and connection errors.
- Nginx Dashboard: Monitors Nginx web server metrics, including request rates and active connections.
- Nomad Dashboard: Visualizes metrics for HashiCorp Nomad.
- PostgreSQL Dashboard: Monitors PostgreSQL performance metrics.
- RabbitMQ Dashboard: Tracks RabbitMQ metrics like queue sizes and message throughput.
- Temporal.io Dashboard: Monitors Temporal.io workflow metrics.
- LLM Observability Dashboard: Visualizes metrics for monitoring large language models.
- SigNoz Ingestion Analysis: Visualizes the volume the metrics, traces and logs ingested into SigNoz. Useful for cost optimization.
- KEDA Dashboard: Monitors metrics for KEDA, a Kubernetes-based event-driven autoscaling component.

LET’S EXPLORE
SIGNOZ CLOUD
TOGETHER!

(?)

Q&A **SESH**

THANKS

@NGIE-MP

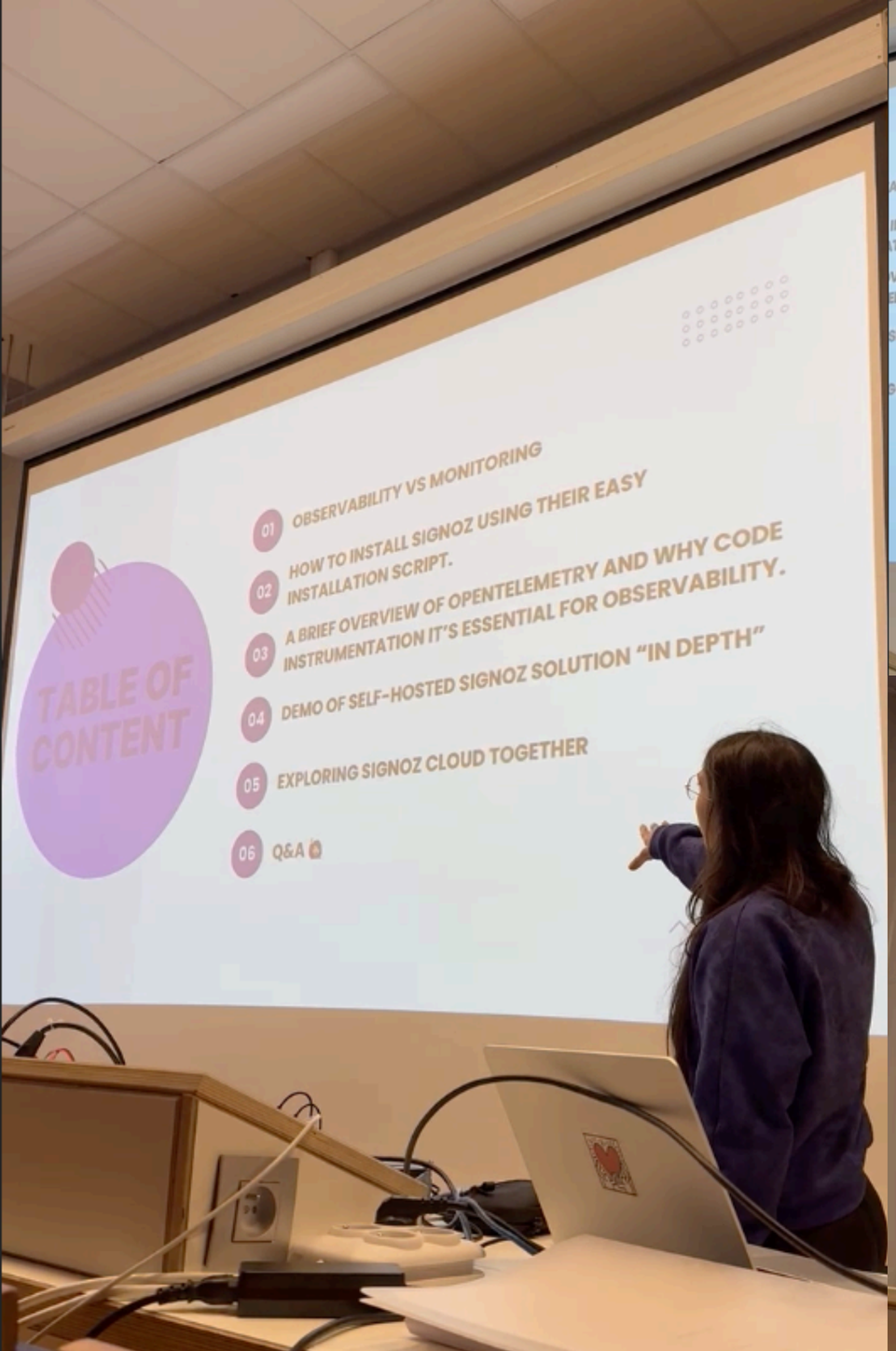
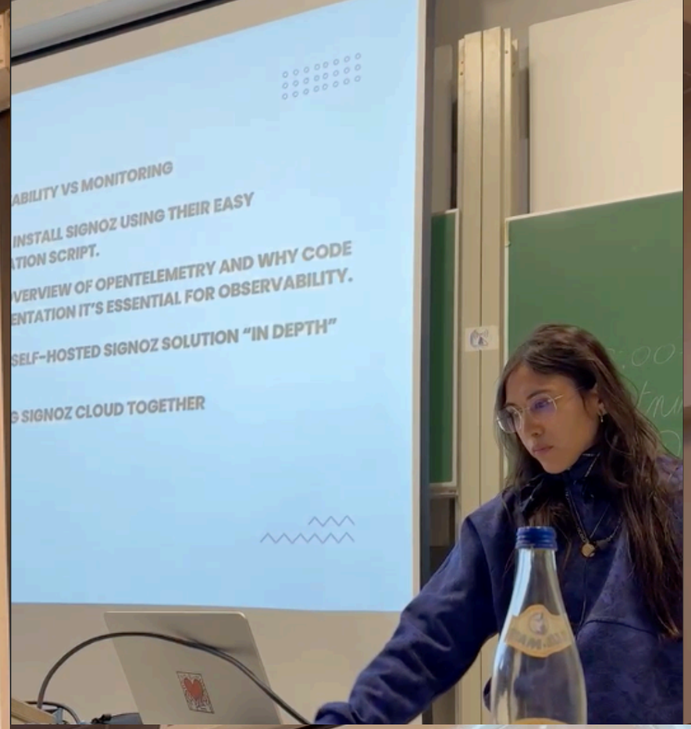
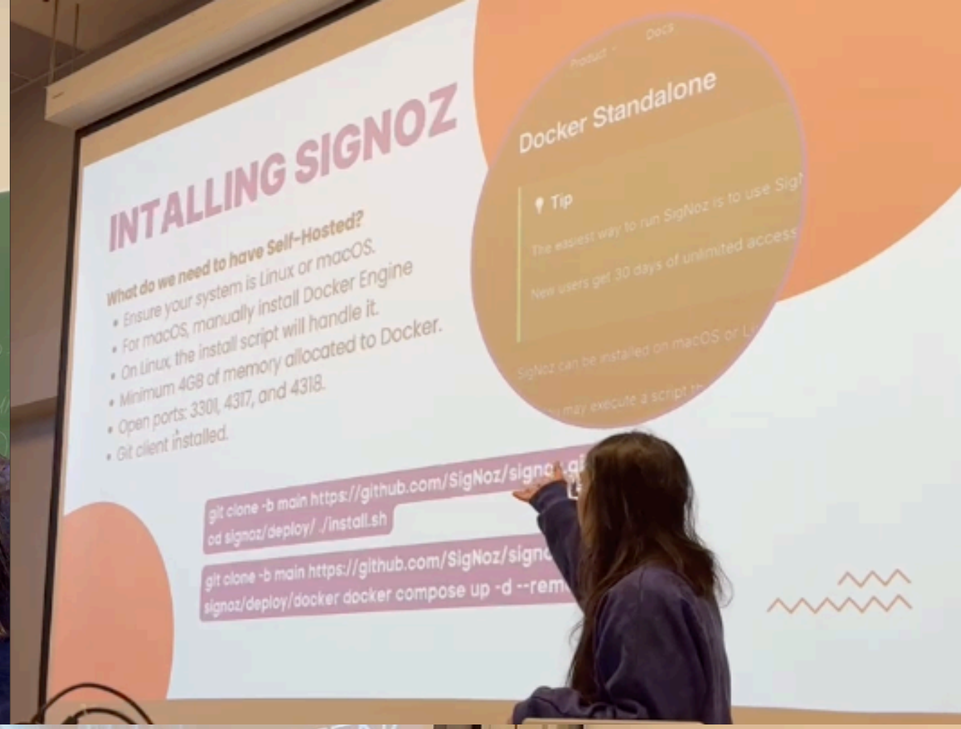
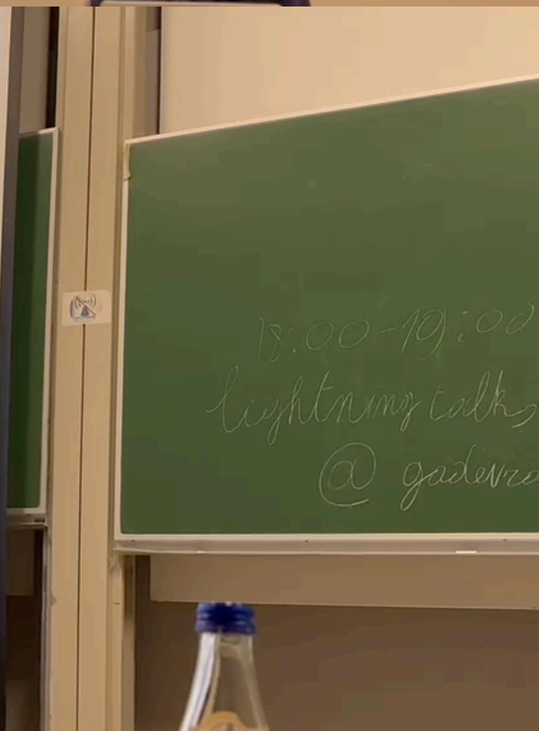


TABLE OF CONTENT
01 OBSERVABILITY VS MONITORING
02 HOW TO INSTALL SIGNOZ USING THEIR EASY INSTALLATION SCRIPT.
03 A BRIEF OVERVIEW OF OPENTELEMETRY AND WHY CODE INSTRUMENTATION IT'S ESSENTIAL FOR OBSERVABILITY.
04 DEMO OF SELF-HOSTED SIGNOZ SOLUTION "IN DEPTH"
05 EXPLORING SIGNOZ CLOUD TOGETHER
06 Q&A


ABILITY VS MONITORING
INSTALL SIGNOZ USING THEIR EASY
ATION SCRIPT.
VERVIEW OF OPENTELEMETRY AND WHY CODE
ENTATION IT'S ESSENTIAL FOR OBSERVABILITY.
SELF-HOSTED SIGNOZ SOLUTION "IN DEPTH"
SIGNOZ CLOUD TOGETHER


INTALLING SIGNOZ
What do we need to have Self-Hosted?
• Ensure your system is Linux or macOS.
• For macOS, manually install Docker Engine
• On Linux, the install script will handle it.
• Minimum 4GB of memory allocated to Docker.
• Open ports: 3301, 4317, and 4318.
• Git client installed.
Docker Standalone
Tip
git clone -b main https://github.com/SigNoz/signoz.git
cd signoz/deploy/ ./install.sh
git clone -b main https://github.com/SigNoz/signoz.git
signoz/deploy/docker docker compose up -d --rem


Imagine Observability Like Cooking
LATENC
& ERROR
ERROR


18:00-19:00
lightning talk
@ godever