AS - Assignment 1

Alexandre Ribeiro - 108122

March 2025

1 Introduction

In modern software development, observability and security are critical components in building reliable and secure applications. This assignment focuses on the integration of OpenTelemetry tracing and security measures within *eShop*, a microservices-based e-Commerce system built using ASP.NET Core.

The primary objective of this work is to instrument a specific feature of the eShop application with OpenTelemetry, enabling end-to-end tracing while ensuring that sensitive data is properly masked or excluded from logs and traces.

2 Selected Feature

Within the application, there are a number of user flows that can be traced and measured to provide important insights into its performance and status. For this assignment, I chose the **User Authentication** feature, which encompasses the login and logout processes. This flow involves several key system components, including the Web App, IdentityAPI, its associated database, and the *IdentityServer* framework (Figure 1).

This feature enables the extraction of valuable insights into user interactions with the application. Key indicators, including login attempts, session

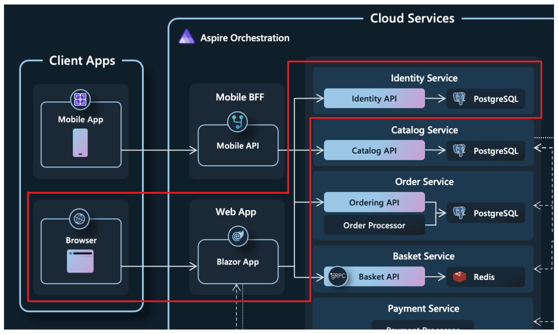


Figure 1: User Authentication Tech Stack

durations, active sessions, and logout events, provide essential data for identifying peak usage periods. This information can be leveraged to optimize resource allocation, ensuring the system efficiently adapts to fluctuations in demand.

The following figures describe the end-to-end interactions between the various system components during the login and logout processes, including the observability tools used to track key metrics (more on observability in Section 3).

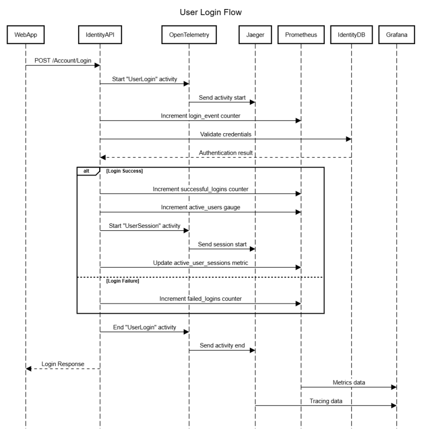


Figure 2: Login Sequence

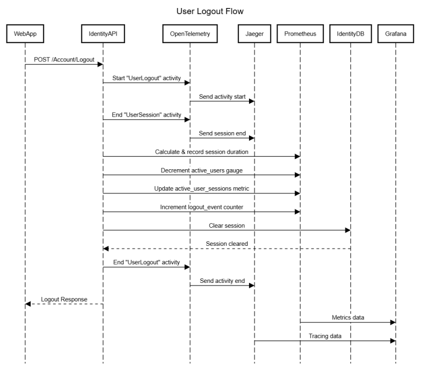


Figure 3: Logout Sequence

3 Observability

The primary objective of this assignment was to integrate observability tools into the system. To achieve this, OpenTelemetry was implemented in the code, along with Jaeger for tracing and Prometheus for metrics collection.The code modifications are located in *src/Identity.API/Quickstart/Account/AccountController.cs*, which contains the relevant API logic for user authentication.

3.1 Traces

To provide end-to-end observability, the following activities are instrumented and traced:

3.1.1 Login Process

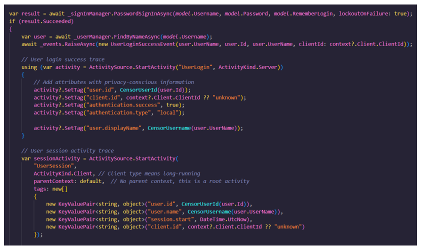
- **Activity: UserLogin**
- **Tags**: user.id (censored), client.id, authentication.success, authentication.type

3.1.2 User Sessions

- **Activity: UserSession**
- **Tags**: user.id (censored), user.name (censored), session.start, client.id
- This is a long-running activity that spans from login to logout.

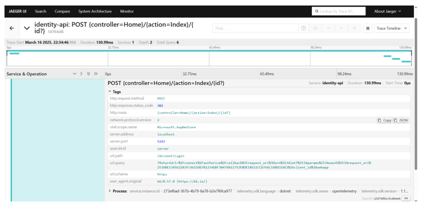
3.1.3 Logout Process

- **Activity: UserLogout**
- **Tags**: user.id (censored), user.displayName (censored), session duration, and timestamps



```
var result = await _signInManager.PasswordSignInAsync(model.Username, model.Password, model.RememberLogin, lockoutOnFailure: true);
if (result.Succeeded)
{
    var user = await _userManager.FindByNameAsync(model.Username);
    await _events.RaiseAsync(new UserLoginSuccessEvent(user.UserName, user.Id, user.UserName, clientId: context?.Client.ClientId));

    // User login success trace
    using (var activity = ActivitySource.StartActivity("UserLogin", ActivityKind.Server))
    {
        // Add attributes with privacy-conscious information
        activity?.SetTag("user.id", CensorUserId(user.Id));
        activity?.SetTag("client.id", context?.Client.ClientId ?? "unknown");
        activity?.SetTag("authentication.success", true);
        activity?.SetTag("authentication.type", "local");

        activity?.SetTag("user.displayName", CensorUsername(user.UserName));
    }

    // User session activity trace
    var sessionActivity = ActivitySource.StartActivity(
        "UserSession",
        ActivityKind.Client, // Client type means long-running
        parentContext: default,  // No parent context, this is a root activity
        tags: new[]
        {
            new KeyValuePair<string, object>("user.id", CensorUserId(user.Id)),
            new KeyValuePair<string, object>("user.name", CensorUsername(user.UserName)),
            new KeyValuePair<string, object>("session.start", DateTime.UtcNow),
            new KeyValuePair<string, object>("client.id", context?.Client.ClientId ?? "unknown")
        });
```

Figure 4: Trace Implementation In Code



identity-api: POST {controller=Home}/{action=Index}/{id?} 14194d6

Trace Start March 16 2025, 22:34:46.956 | Duration 130.99ms | Services 1 | Depth 2 | Total Spans 6

0µs 32.75ms 65.49ms 98.24ms 130.99ms

POST {controller=Home}/{action=Index}/{id?} Service: identity-api | Duration: 130.99ms | Start Time: 0µs

Tags

http.request.method	POST
http.response.status_code	302
http.route	{controller=Home}/{action=Index}/{id?}
network.protocol.version	2
otel.scope.name	Microsoft.AspNetCore
server.address	localhost
server.port	5243
span.kind	server
url.path	/Account/Login
url.query	?ReturnUrl=%2Fconnect%2Fauthorize%2Fcallback%3Frequest_uri%3Durn%253Aietf%253Aparams%253Aoauth%253Arequest_uri%253ABE134562283FC4A15DD7B127A6BF30474811752E0DE10521CCD766C16BDCEAE2%26client_id%3Dwebapp
url.scheme	https
user_agent.original	k6/0.57.0 (https://k6.io/)

Process: service.instance.id = 273ef6ad-367b-4b79-8a78-b2e7f6fca977 | telemetry.sdk.language = dotnet | telemetry.sdk.name = opentelemetry | telemetry.sdk.version = 1.1...

SpanID: c2d7d46cc3cdbeeb

Figure 5: Login Trace Example In Jaeger

3.2 Metrics

To monitor the authentication flow, several key metrics are collected and analyzed:

3.2.1 Active Users

- **identity_active_users**: Represents the number of currently logged-in users.
- **identity_active_user_sessions**: Provides detailed information about active user sessions.

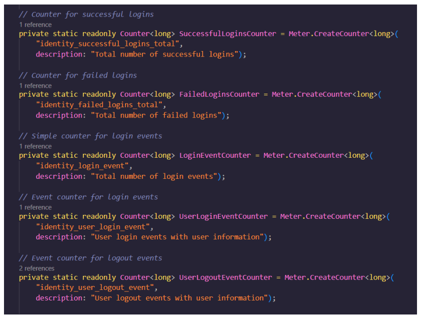
3.2.2 Authentication Events

- **identity_successful_logins_total**: A counter tracking the total number of successful login attempts.
- **identity_failed_logins_total**: A counter tracking the total number of failed login attempts.

- **identity_login_event**: A simple counter recording login events.
- **identity_user_login_event**: A detailed event log including user-specific information for each login.
- **identity_user_logout_event**: A detailed event log including user-specific information for each logout.

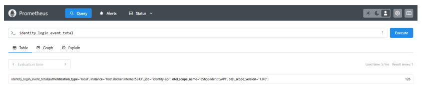
3.2.3 Session Analytics

- **identity_session_duration_seconds**: A histogram capturing the distribution of user session durations.



```
// Counter for successful logins
1 reference
private static readonly Counter<long> SuccessfulLoginsCounter = Meter.CreateCounter<long>(
    "identity_successful_logins_total",
    description: "Total number of successful logins");

// Counter for failed logins
1 reference
private static readonly Counter<long> FailedLoginsCounter = Meter.CreateCounter<long>(
    "identity_failed_logins_total",
    description: "Total number of failed logins");

// Simple counter for login events
1 reference
private static readonly Counter<long> LoginEventCounter = Meter.CreateCounter<long>(
    "identity_login_event",
    description: "Total number of login events");

// Event counter for login events
1 reference
private static readonly Counter<long> UserLoginEventCounter = Meter.CreateCounter<long>(
    "identity_user_login_event",
    description: "User login events with user information");

// Event counter for logout events
2 references
private static readonly Counter<long> UserLogoutEventCounter = Meter.CreateCounter<long>(
    "identity_user_logout_event",
    description: "User logout events with user information");
```

Figure 6: Metric Implementation In Code

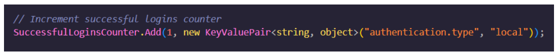


```
identity_login_event_total
```

Load time: 57ms   Result series: 1

```
identity_login_event_total{authentication_type="local", instance="host.docker.internal:5243", job="identity-api", otel_scope_name="eShop.IdentityAPI", otel_scope_version="1.0.0"}   126
```

Figure 7: Login Event Metric Example In Prometheus

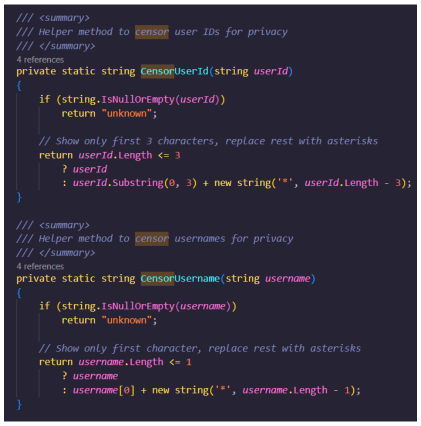


```
// Increment successful logins counter
SuccessfulLoginsCounter.Add(1, new KeyValuePair<string, object>("authentication.type", "local"));
```

Figure 8: Counter Increment Example

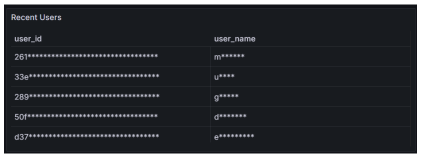
4 Data Scrubbing

In the traces, there are some tags that contain sensitive user information, particularly their id and name. In order to hide these details from unauthorized access, a data scrubbing technique was used in which only the first characters (one in the name and three in the id) are shown while the rest are replaced with asterisks.



```
/// <summary>
/// Helper method to censor user IDs for privacy
/// </summary>
4 references
private static string CensorUserId(string userId)
{
    if (string.IsNullOrEmpty(userId))
        return "unknown";

    // Show only first 3 characters, replace rest with asterisks
    return userId.Length <= 3
        ? userId
        : userId.Substring(0, 3) + new string('*', userId.Length - 3);
}

/// <summary>
/// Helper method to censor usernames for privacy
/// </summary>
4 references
private static string CensorUsername(string username)
{
    if (string.IsNullOrEmpty(username))
        return "unknown";

    // Show only first character, replace rest with asterisks
    return username.Length <= 1
        ? username
        : username[0] + new string('*', username.Length - 1);
}
```

Figure 9: Data Scrubbing Implementation In Code



Recent Users

user_id	user_name
261*********************************	m******
33e*********************************	u****
289*********************************	g*****
50f*********************************	d*******
d37*********************************	e*********

Figure 10: Data Scrubbing Result

5 Grafana Dashboards

In order to visualize the collected traces and metrics, a dashboard was developed using Grafana containing relevant information regarding the system status.

The visualizations show:

- Number of current active users
- Average user session duration
- Id and name of users that recently logged in
- Number of successful login attempts
- Number of failed login attempts
- Total number of login attempts
- Trace data, regarding login, logout and other requests going in or out of the Identity API

Another dashboard was also imported to showcase ASP.Net Core data, which is a template downloaded from https://grafana.com/grafana/dashboards/19924-asp-net-core/. It shows basic information regarding requests and connections inside the system.

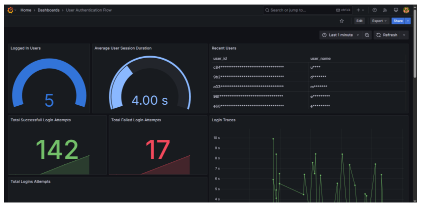

Logged in Users
5
Average User Session Duration
4.00 s
Recent Users
Total Successfull Login Attempts
142
Total Failed Login Attempts
17
Login Traces
Total Logins Attempts

Figure 11: User Authentication Dashboard - 1

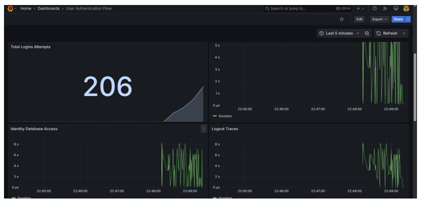


Figure 12: User Authentication Dashboard - 2

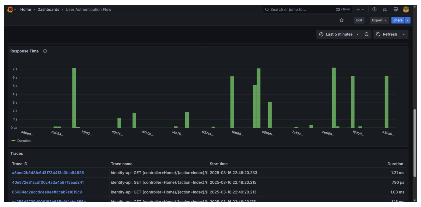


Figure 13: User Authentication Dashboard - 3

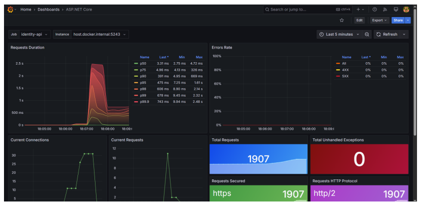


Figure 14: ASP.Net Dashboard

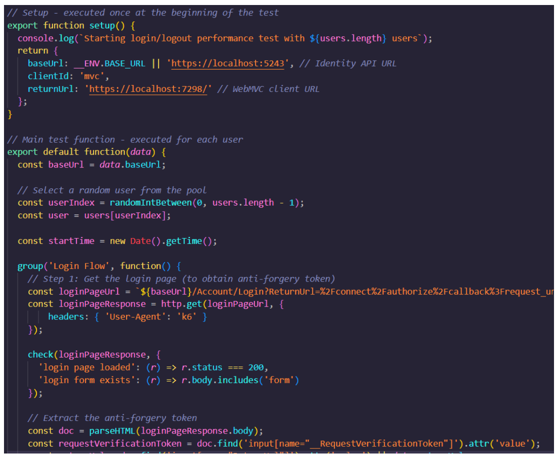
6 Load Testing

Testing user login and logout by hand is a tiresome job, so, to automate the process, k6 was used to load test the application.

In the original repository, only two users are pre-defined in the system, *bob* and *alice*, which is not optimal for tests that aim to simulate higher workloads. Because of that, 19 other users were created, all with the same password for simplicity. Additionally, the script tests three of those users using wrong credentials, to evoke failed login attempts.

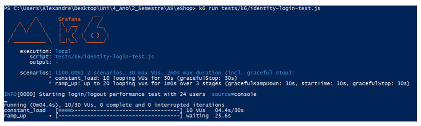
The script tests for two scenarios:

- **Constant Load:** 10 virtual users for 30 seconds
- **Ramp Up**: Starting with 0 users, ramping up to 20 users, then back down to 0



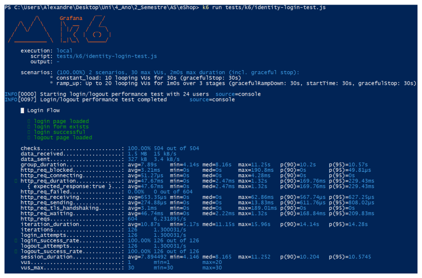
```javascript
// Setup - executed once at the beginning of the test
export function setup() {
  console.log(`Starting login/logout performance test with ${users.length} users`);
  return {
    baseUrl: __ENV.BASE_URL || 'https://localhost:5243', // Identity API URL
    clientId: 'mvc',
    returnUrl: 'https://localhost:7298/' // WebMVC client URL
  };
}

// Main test function - executed for each user
export default function(data) {
  const baseUrl = data.baseUrl;

  // Select a random user from the pool
  const userIndex = randomIntBetween(0, users.length - 1);
  const user = users[userIndex];

  const startTime = new Date().getTime();

  group('Login Flow', function() {
    // Step 1: Get the login page (to obtain anti-forgery token)
    const loginPageUrl = `${baseUrl}/Account/Login?ReturnUrl=%2Fconnect%2Fauthorize%2Fcallback%3Frequest_ur
    const loginPageResponse = http.get(loginPageUrl, {
        headers: { 'User-Agent': 'k6' }
    });

    check(loginPageResponse, {
      'login page loaded': (r) => r.status === 200,
      'login form exists': (r) => r.body.includes('form')
    });

    // Extract the anti-forgery token
    const doc = parseHTML(loginPageResponse.body);
    const requestVerificationToken = doc.find('input[name="__RequestVerificationToken"]').attr('value');
```

Figure 15: K6 Script Excerpt



```
PS C:\Users\Alexandre\Desktop\Uni\4_Ano\2_Semestre\AS\eShop> k6 run tests/k6/identity-login-test.js

         Grafana

     execution: local
        script: tests/k6/identity-login-test.js
        output: -

     scenarios: (100.00%) 2 scenarios, 30 max VUs, 2m0s max duration (incl. graceful stop):
              * constant_load: 10 looping VUs for 30s (gracefulStop: 30s)
              * ramp_up: Up to 20 looping VUs for 1m0s over 3 stages (gracefulRampDown: 30s, startTime: 30s, gracefulStop: 30s)

INFO[0000] Starting login/logout performance test with 24 users  source=console

running (0m04.4s), 10/30 VUs, 0 complete and 0 interrupted iterations
constant_load   [====>-------------------------------] 10 VUs   04.4s/30s
ramp_up       • [------------------------------------] waiting  25.6s
```

Figure 16: Load Testing In Progress



```
PS C:\Users\Alexandre\Desktop\Uni\4_Ano\2_Semestre\AS\eShop> k6 run tests/k6/identity-login-test.js

         Grafana

     execution: local
        script: tests/k6/identity-login-test.js
        output: -

     scenarios: (100.00%) 2 scenarios, 30 max VUs, 2m0s max duration (incl. graceful stop):
              * constant_load: 10 looping VUs for 30s (gracefulStop: 30s)
              * ramp_up: Up to 20 looping VUs for 1m0s over 3 stages (gracefulRampDown: 30s, startTime: 30s, gracefulStop: 30s)

INFO[0000] Starting login/logout performance test with 24 users  source=console
INFO[0097] Login/logout performance test completed       source=console

     █ Login Flow

       □ login page loaded
       □ login form exists
       □ login successful
       □ logout page loaded

     checks.........................: 100.00% 504 out of 504
     data_received..................: 1.5 MB  15 kB/s
     data_sent......................: 327 kB  3.4 kB/s
     group_duration.................: avg=7.89s    min=4.14s med=8.16s  max=11.25s   p(90)=10.2s    p(95)=10.57s
     http_req_blocked...............: avg=3.21ms   min=0s    med=0s     max=190.8ms  p(90)=0s       p(95)=49.81µs
     http_req_connecting............: avg=51.27µs  min=0s    med=0s     max=4.28ms   p(90)=0s       p(95)=0s
   □ http_req_duration..............: avg=47.67ms  min=0s    med=2.47ms max=1.32s    p(90)=169.76ms p(95)=229.43ms
       { expected_response:true }...: avg=47.67ms  min=0s    med=2.47ms max=1.32s    p(90)=169.76ms p(95)=229.43ms
     http_req_failed................: 0.00%   0 out of 604
     http_req_receiving.............: avg=653.35µs min=0s    med=0s     max=62.86ms  p(90)=567.74µs p(95)=627.25µs
     http_req_sending...............: avg=274.88µs min=0s    med=0s     max=13.83ms  p(90)=611.76µs p(95)=808.02µs
     http_req_tls_handshaking.......: avg=3.1ms    min=0s    med=0s     max=189.01ms p(90)=0s       p(95)=0s
     http_req_waiting...............: avg=46.74ms  min=0s    med=2.22ms max=1.32s    p(90)=168.84ms p(95)=209.83ms
     http_reqs......................: 604     6.231895/s
     iteration_duration.............: avg=10.87s   min=5.17s med=11.15s max=15.96s   p(90)=14.14s   p(95)=14.28s
     iterations.....................: 126     1.300031/s
     login_attempts.................: 126     1.300031/s
   □ login_success_rate.............: 100.00% 126 out of 126
     logout_attempts................: 126     1.300031/s
   □ logout_success_rate............: 100.00% 126 out of 126
     session_duration...............: avg=7.894492 min=4.146 med=8.165  max=11.252   p(90)=10.204   p(95)=10.5745
     vus............................: 1       min=1         max=20
     vus_max........................: 30      min=30        max=30
```

Figure 17: Load Testing Results

7 Usage of Gen AI

In this assignment, there were a number of instances in which generative AI tools were used to perform tasks or help write and fix code. The particular models used included GPT-4o (and GPT-4o mini when token limits were reached) to generate insights, explanations, and code suggestions. Additionally, Claude 3.7 Sonnet was leveraged through GitHub Copilot to assist with code completion, debugging, and refining implementation details.

These AI tools played an important role in accelerating development; however, they served as a complement to detailed research, which was essential to understand the underlying concepts and ensure their correct implementation.

8 Challenges

This assignment, while simple in nature, presented some challenges that made its difficulty higher than expected. The programming language, for starters, was the initial barrier in my development, due to the inexperience I had with C#, and by extension the .NET framework. It took a number of hours to fully understand the structure of the project, the interactions between the many components, what code needed to be targeted for the assignment and what configurations needed to be changed/added.

Implementing the observability tools was initially challenging due to outdated documentation and unclear instructions regarding the appropriate packages to use. However, once the setup was functional, these issues no longer posed a problem.

9 Conclusion

This assignment provided a valuable opportunity to implement observability tools within a microservices architecture. By integrating OpenTelemetry, Jaeger, and Prometheus into the eShop authentication flow, I created a monitoring system that offers insights into user behavior and system performance. Working with these tools required overcoming technical challenges, particularly given my limited prior experience with C# and the .NET ecosystem. This constraint ultimately expanded my technical knowledge as I navigated the project structure and implemented the necessary changes.

As for accomplishments, I successfully instrumented the authentication flow with traces that track the user journey from login to logout while implementing data scrubbing to protect sensitive information. The metrics collection now provides visibility into key statistics, and the custom Grafana dashboards transform telemetry into actionable visualizations. The k6 load testing validated both the application and observability infrastructure under various user loads.

10 Link To Repository

All of my work can be found in my GitHub repository (`https://github.com/Sytuz/eShop`), which is a fork from the original eShop repository.