**Хід роботи**

Бекенд застосунку комунікує з Binance Websocket та повертає інформація про деякі монети до авторизованого користувача. Якщо користувач неавторизований – він отримає помилку, якщо захоче подивитися на дані про криптовалюти.

Клас BinanceStreamListener відповідає за підключення до Binance WebSocket щоб отримувати в реальному часі інформацію про задану криптовалюту. Він приймає дані з біржі і передає їх далі для обробки.

Клас BinanceStreamManager керує усіма об’єктами типу BinanceStreamListener.

Клас CryptoWebSocketHandler відповідає за обробку WebSocket з’єднання від клієнта, передає кожну секунду інформацію, отриману від біржі до користувача. Користувач отримує назву криптовалюти, її поточну ціну та останній час оновлення.

Клас WebSocketController відповідає за установлення WebSocket з’єднання з користувачем. Замість метода GET він очікує метод CONNECT через [HTTP/2](https://learn.microsoft.com/en-us/aspnet/core/fundamentals/websockets?view=aspnetcore-9.0).

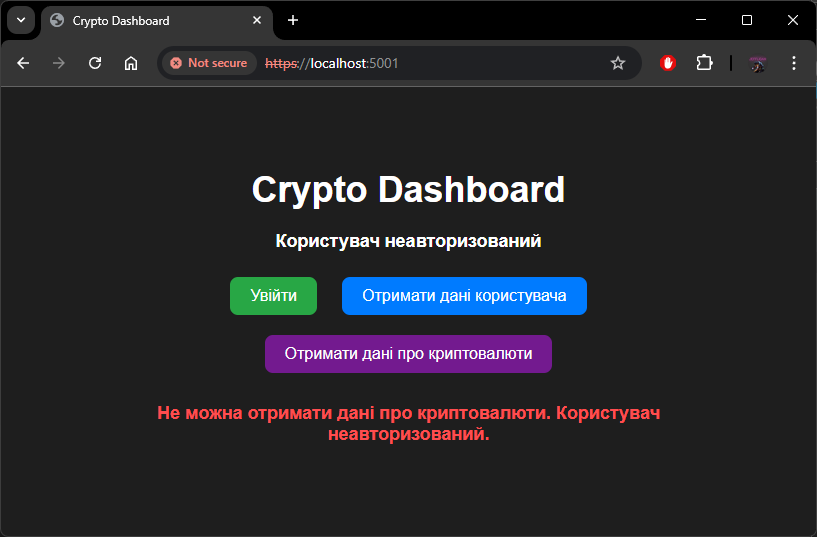


Рисунок 4.1 – Помилка, яка виникає, якщо користувач хоче отримати дані про криптовалюти без авторизації

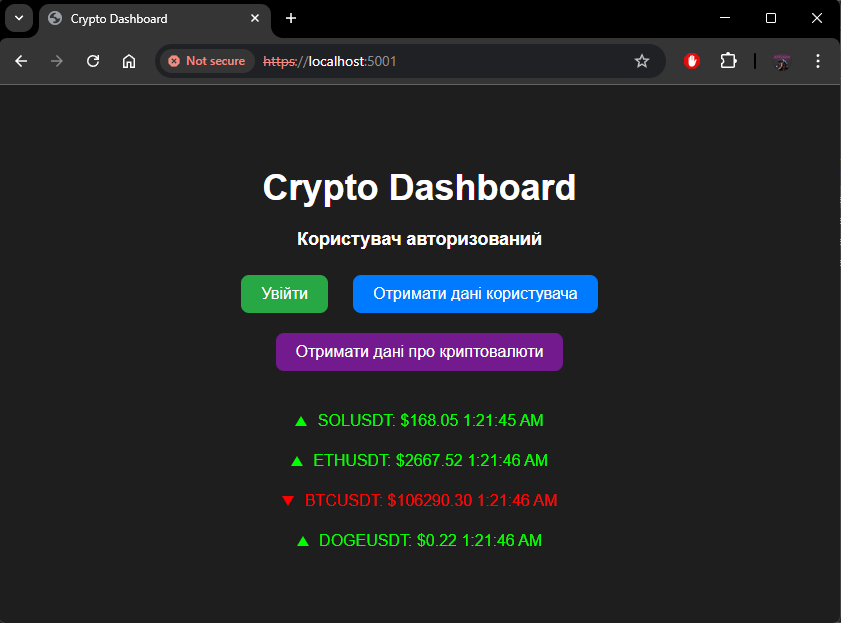


Рисунок 4.2 – Інформація про криптовалюти у реальному часі

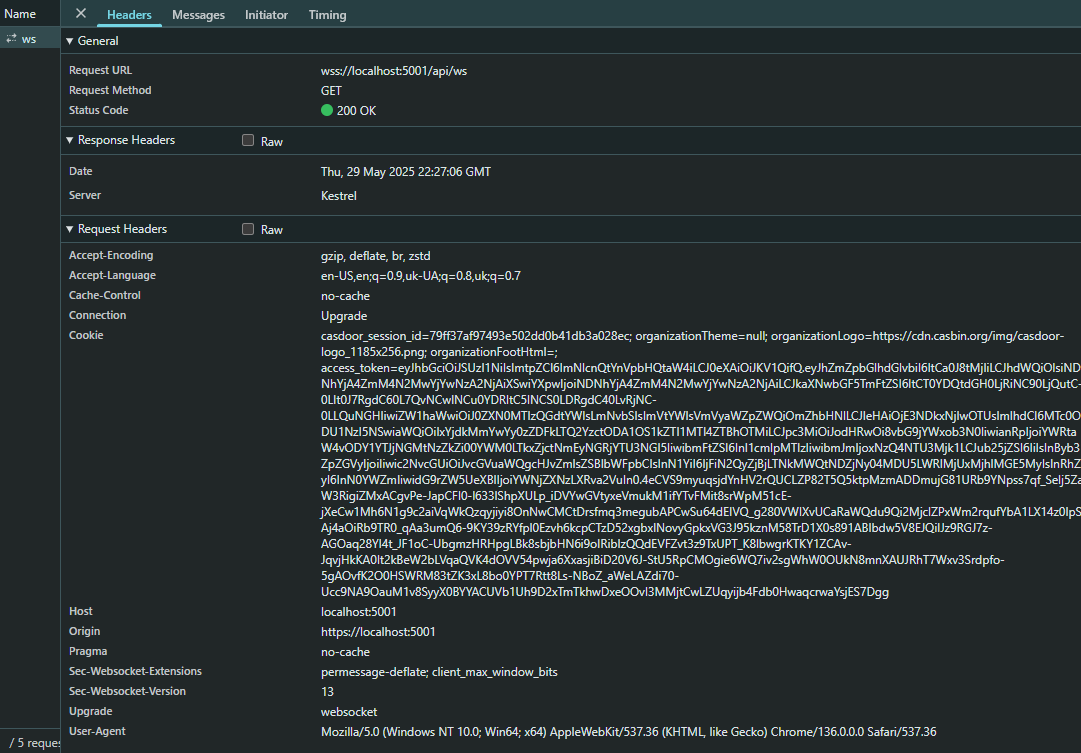


Рисунок 4.3 – Заголовки запиту на створення WebSocket з’єднання

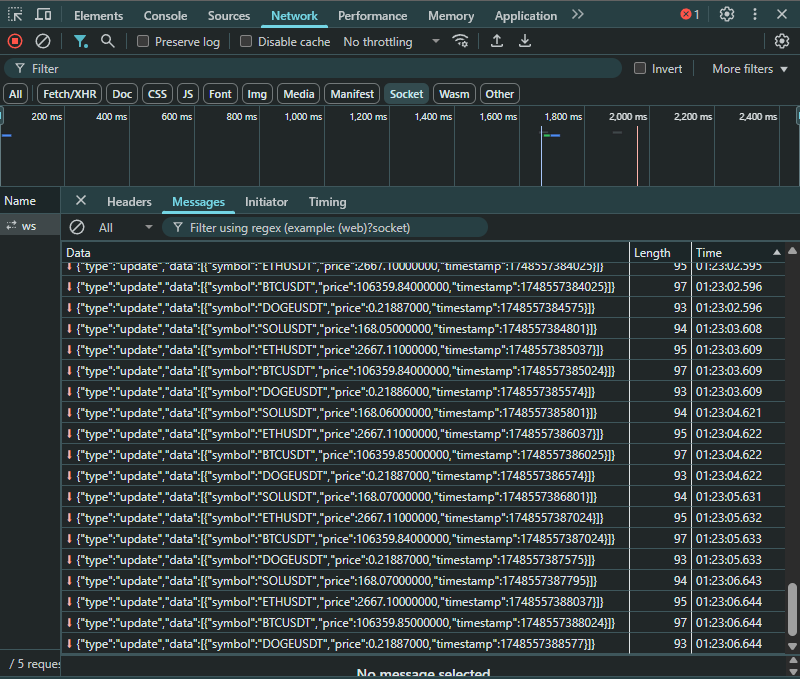


Рисунок 4.4 – Повідомлення, які надходять до клієнта по WebSocket у консолі

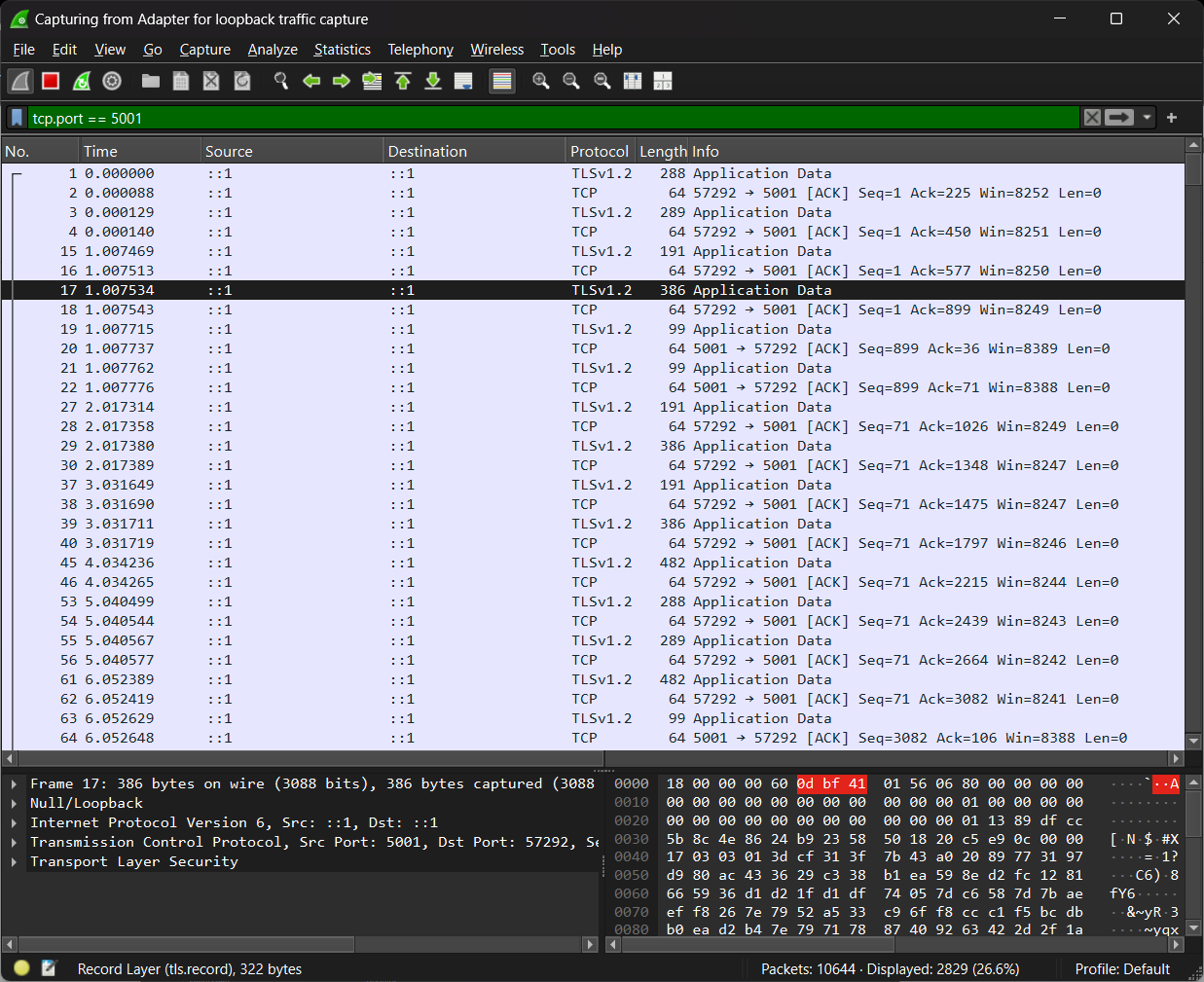


Рисунок 4.5 – Wireshark показує, що комунікація відбувається по TLS v1.2

**Лістинг 4.1. Клас BinanceStreamListener**

using System.Net.WebSockets;

using System.Text;

using System.Threading.Channels;

public class BinanceStreamListener

{

private readonly string \_symbol;

private readonly Channel<string> \_channel;

private readonly CancellationToken \_token;

public BinanceStreamListener(string symbol, Channel<string> channel, CancellationToken token)

{

\_symbol = symbol;

\_channel = channel;

\_token = token;

}

public async Task StartAsync()

{

var url = new Uri($"wss://stream.binance.com:9443/ws/{\_symbol}@ticker");

using var ws = new ClientWebSocket();

try

{

await ws.ConnectAsync(url, \_token);

Console.WriteLine($"Connected to Binance stream for {\_symbol}");

var buffer = new byte[8192];

while (!\_token.IsCancellationRequested && ws.State == WebSocketState.Open)

{

var result = await ws.ReceiveAsync(buffer, \_token);

if (result.MessageType == WebSocketMessageType.Close)

break;

var json = Encoding.UTF8.GetString(buffer, 0, result.Count);

await \_channel.Writer.WriteAsync(json, \_token);

}

}

catch (Exception ex)

{

Console.WriteLine($"Error in Binance stream for {\_symbol}: {ex.Message}");

}

finally

{

\_channel.Writer.Complete();

}

}

}

**Лістинг 4.2. Клас BinanceStreamManager**

using System.Collections.Concurrent;

using System.Threading.Channels;

public class BinanceStreamManager

{

private readonly List<string> \_defaultSymbols = new() { "btcusdt", "ethusdt", "solusdt", "dogeusdt" };

private readonly ConcurrentDictionary<string, Channel<string>> \_tracked = new();

private readonly CancellationTokenSource \_cts = new();

public BinanceStreamManager()

{

foreach (var sym in \_defaultSymbols)

{

var ch = Channel.CreateUnbounded<string>();

\_tracked[sym] = ch;

var listener = new BinanceStreamListener(sym, ch, \_cts.Token);

\_ = listener.StartAsync();

}

}

public IReadOnlyDictionary<string, Channel<string>> GetTrackedStreams() => \_tracked;

public void CancelAll() => \_cts.Cancel();

}

**Лістинг 4.3. Клас CryptoWebSocketHandler**

using System.Collections.Concurrent;

using System.Net.WebSockets;

using System.Text;

using System.Text.Json;

public class CryptoWebSocketHandler

{

private readonly BinanceStreamManager \_streamManager;

public CryptoWebSocketHandler(BinanceStreamManager streamManager)

{

\_streamManager = streamManager;

}

public async Task HandleSocketAsync(WebSocket socket)

{

var streams = \_streamManager.GetTrackedStreams();

var latest = new ConcurrentDictionary<string, string>();

var cts = new CancellationTokenSource();

foreach (var kvp in streams)

{

var symbol = kvp.Key;

var channel = kvp.Value;

\_ = Task.Run(async () =>

{

try

{

await foreach (var msg in channel.Reader.ReadAllAsync(cts.Token))

{

latest[symbol] = msg;

}

}

catch (OperationCanceledException)

{

}

catch (Exception ex)

{

Console.WriteLine($"Error reading from channel for {symbol}: {ex.Message}");

}

});

}

var buffer = new byte[1024];

var sendInterval = TimeSpan.FromSeconds(1);

try

{

while (socket.State == WebSocketState.Open)

{

var receiveTask = socket.ReceiveAsync(new ArraySegment<byte>(buffer), CancellationToken.None);

var delayTask = Task.Delay(0);

var completed = await Task.WhenAny(receiveTask, delayTask);

if (completed == receiveTask)

{

var result = receiveTask.Result;

if (result.MessageType == WebSocketMessageType.Close)

{

await socket.CloseAsync(WebSocketCloseStatus.NormalClosure,

"Client closed",

CancellationToken.None);

break;

}

}

foreach (var sym in latest.Keys)

{

try

{

var doc = JsonDocument.Parse(latest[sym]);

var root = doc.RootElement;

var price = decimal.Parse(root.GetProperty("c").GetString()!);

var ts = root.GetProperty("E").GetInt64();

var dataObj = new

{

type = "update",

data = new[]

{

new {

symbol = sym.ToUpper(),

price,

timestamp = ts

}

}

};

var bytes = Encoding.UTF8.GetBytes(JsonSerializer.Serialize(dataObj));

await socket.SendAsync(bytes, WebSocketMessageType.Text, true, CancellationToken.None);

}

catch { }

}

await Task.Delay(sendInterval);

}

}

catch (WebSocketException) { }

catch (OperationCanceledException) { }

finally

{

cts.Cancel();

if (socket.State != WebSocketState.Closed)

{

try

{

await socket.CloseAsync(WebSocketCloseStatus.NormalClosure,

"Cleaning up",

CancellationToken.None);

}

catch { }

}

}

}

}

**Лістинг 4.4. Клас WebSocketController**

using Microsoft.AspNetCore.Mvc;

using Microsoft.Extensions.Options;

[ApiController]

public class WebSocketController : ControllerBase

{

private readonly OAuthOptions \_oauthOptions;

private readonly CryptoWebSocketHandler \_handler;

public WebSocketController(CryptoWebSocketHandler handler, IOptions<OAuthOptions> options)

{

\_handler = handler;

\_oauthOptions = options.Value;

}

[Route("/api/ws/crypto")]

public async Task Get()

{

if (!HttpContext.WebSockets.IsWebSocketRequest)

{

HttpContext.Response.StatusCode = 400;

return;

}

if (!Request.Cookies.TryGetValue("access\_token", out var accessToken) || string.IsNullOrEmpty(accessToken))

{

HttpContext.Response.StatusCode = 401;

return;

}

try

{

var payloadJson = JwtDecoder.DecodeJwtPayload(accessToken);

using var httpClient = new HttpClient();

var response = await httpClient.GetAsync(\_oauthOptions.JwtVerifyEndpoint);

var jwks = await response.Content.ReadAsStringAsync();

if (!JwtDecoder.ValidateJwt(accessToken, jwks))

{

Console.WriteLine("Invalid or expired access token according to Casdoor");

HttpContext.Response.StatusCode = 401;

return;

}

}

catch (Exception ex)

{

Console.WriteLine("Error decoding JWT or validating token: " + ex.Message);

HttpContext.Response.StatusCode = 401;

return;

}

using var ws = await HttpContext.WebSockets.AcceptWebSocketAsync();

await \_handler.HandleSocketAsync(ws);

}

}

**Висновки:** розширив програму, створену для 3 лабораторної роботи, додавши комунікацію між клієнтом та сервером з використанням WebSocket, щоб у реальному часі передавати інформацію про криптовалюти, яка отримується за допомогою комунікації між сервером та Binance також з використанням WebSocket.