Microservices & Machine learning with .NET 5

ALTEN SWEDEN

Ahmed Morsi

Azure Solution Architect & .NET Engineer

Stockholm – 17 | 05 | 2021

Who am I

Microsoft Azure Certified Solutions Architect with more than 11 years of experience in the software analysis, design and development.

I help customers solve their business problems through software.
I serve, mentor, and pave the way to teams I lead so that they successfully achieve the goal they are set to achieve.
Currently working with ABB as Microsoft Azure specialist.

Ahmed Morsi
Azure Solutions Architect & .NET Engineer

LinkedIn https://www.linkedin.com/in/eramax
E-mail ahmed.morsi@alten.se
Blog https://emolike.net/

Topics to be covered in this lecture

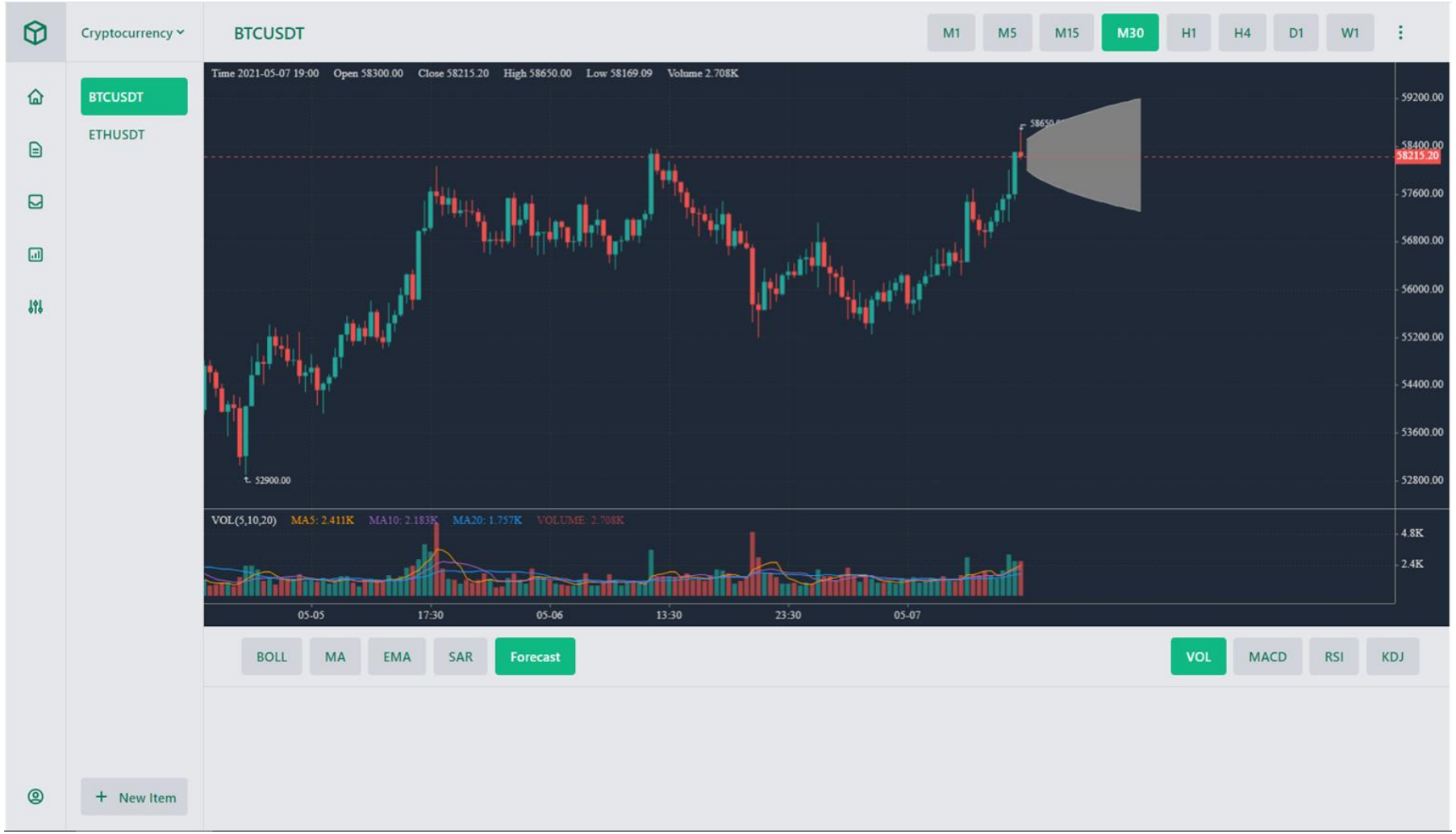
- How to build microservices with .NET
- How to secure your microservice
- How to build machine learning microservice with .NET
- How are microservices communicate each other
- How to use RabbitMQ
- How to stream real time data to clients using SignalR
- How to build Vue front-end application
- And more!

Demo Application

What is the best idea to develop a simple application to cover

1. Microservice architecture
2. SignalR real time messaging
3. RabbitMQ service bus
4. Secured microservices
5. Machine learning
6. Nice front end

Demo Application



https://github.com/eramax/CryptoPredictor

Demo Application

What are microservices?

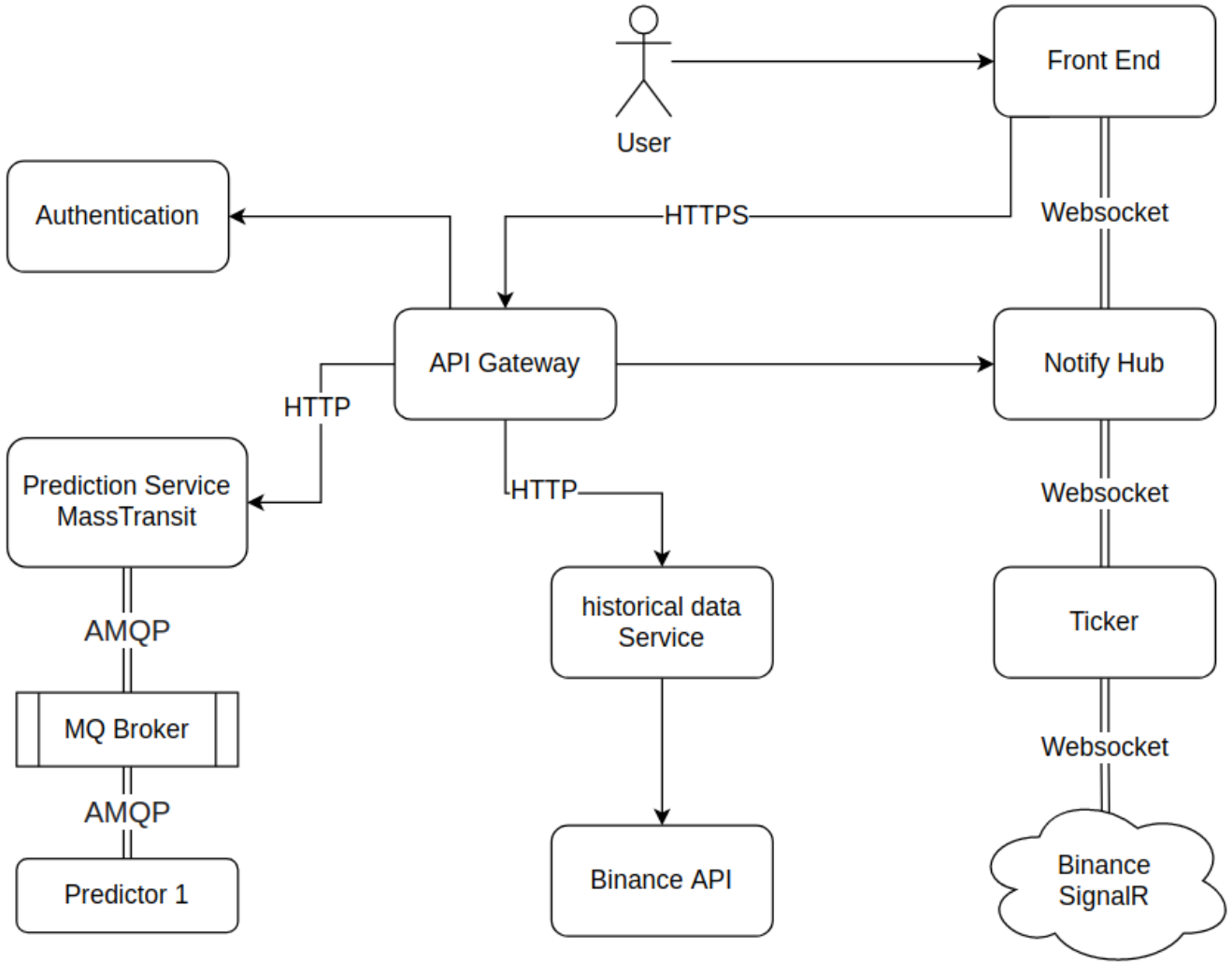
A microservice is a service with one, and only one, very narrowly focused capability that a remote API exposes to the rest of the system.

It is an approach to developing a single application as a suite of small services, each running in its own process and communicating with lightweight mechanisms, often an HTTP resource API, Service Bus, etc.

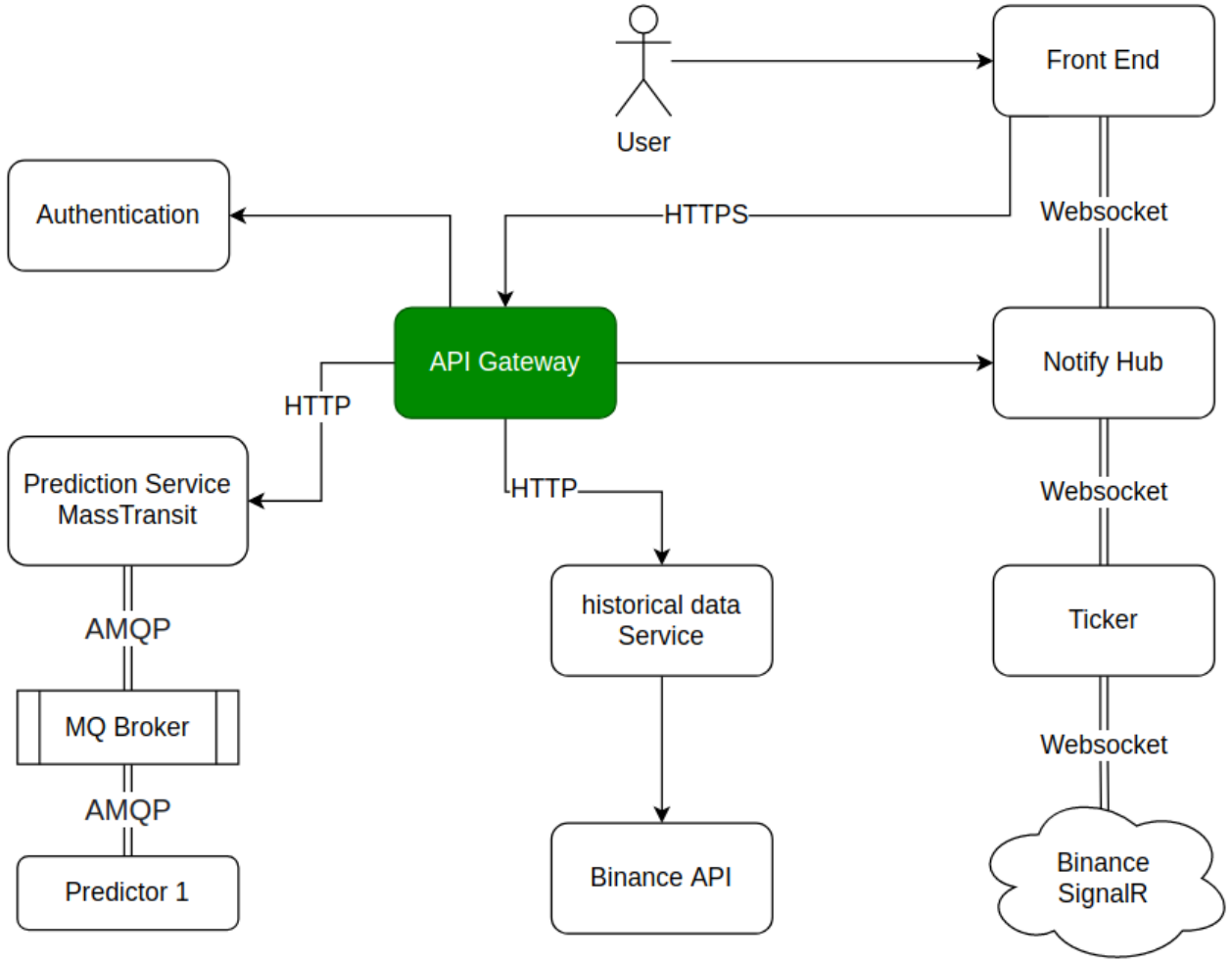
Microservice characteristics

- Single Responsibility for capability
- Individually Deployable
- Consists of One or More processes
- Have its own data source
- Is replaceable
- Can be maintained by small teams

Demo Application



1- API Gateway Ocelot microservice



1- API Gateway Ocelot microservice

What it does

- Expose list of endpoint to the client to allow the client to reach all other microservices behind the API gateway using only API gateway domain
- Route all the requests to the desired microservice.
- Validate Authentication before routing requests
- Enable caching for microservices responses
- Enable retry policies to reach the microservices
- Enable circuit breaker for unavailable microservices

How it works

- It uses a json configuration file to be configured to do all above tasks

NuGet packages

- Ocelot
- Ocelot.Provider.Polly
- Ocelot.Cache.CacheManager

API Gateway definition

API gateway that is the single-entry point for all clients. The API gateway handles requests in one of two ways. Some requests are simply proxied/routed to the appropriate service. It handles other requests by fanning out to multiple services.

Solve issues like

- **Coupling**: Without the API Gateway pattern the client apps are coupled to the internal microservices.
- **Too many round-trips:** A single page/screen in the client app might require several calls to multiple services.
- **Security issues:** Without a gateway, all the microservices must be exposed to the “external world”
- **Cross-cutting concerns:** Each publicly published microservice must handle concerns such as authorization, SSL, etc.

Ocelot

Ocelot is an open-source .NET Core based API Gateway especially made for microservices architecture that need unified points of entry into their system. It is lightweight, fast, scalable and provides routing and authentication among many other features.

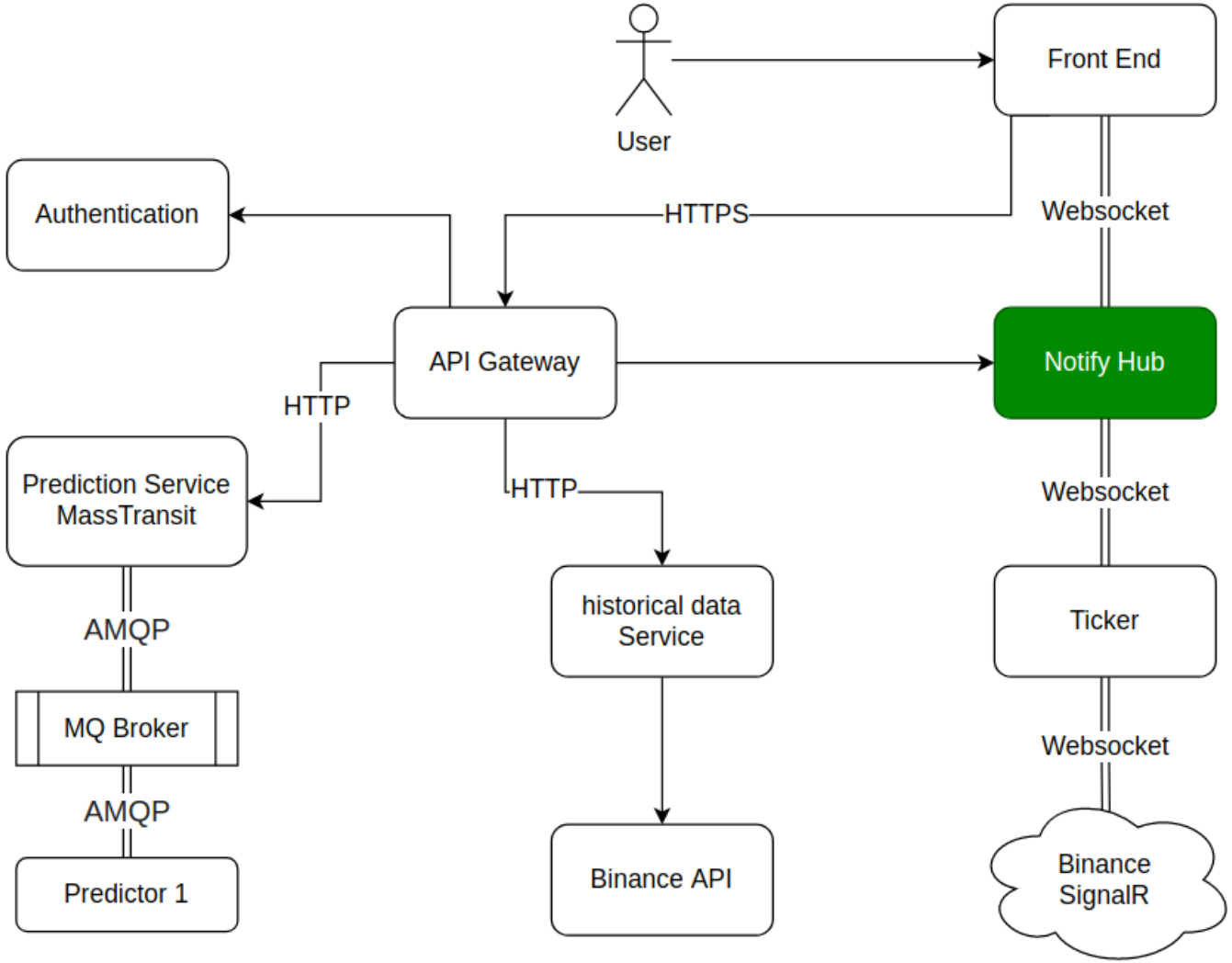
Benefits

- Authentication and authorization
- Service discovery integration
- Response caching
- Retry policies, circuit breaker and QoS
- Rate limiting and throttling
- Load balancing
- Logging, tracing, correlation
- Headers, query strings and claims transformation
- IP whitelisting

API Gateway Main

```
string stringKey = "";
Host.CreateDefaultBuilder(args)
    .ConfigureLogging(logging => logging.AddConsole())
    .ConfigureWebHostDefaults(webBuilder => webBuilder.UseUrls("https://0.0.0.0:5000")
    .ConfigureAppConfiguration((hostingContext, config) => stringKey = config.Build().GetValue<string>("Secret"))
    .ConfigureServices(services =>
    {
        var key = Encoding.UTF8.GetBytes(stringKey);
        services.AddCors().AddOcelot().AddCacheManager(settings => settings.WithDictionaryHandle()).AddPolly();
        services.AddAuthentication(x => x.DefaultAuthenticateScheme = x.DefaultSignInScheme = JwtBearerDefaults.AuthenticationScheme)
            .AddJwtBearer(x =>
            {
                x.RequireHttpsMetadata = false;
                x.SaveToken = true;
                x.TokenValidationParameters = new TokenValidationParameters
                {
                    ValidateIssuerSigningKey = true,
                    IssuerSigningKey = new SymmetricSecurityKey(key),
                    ValidateIssuer = false,
                    ValidateAudience = false
                };
            });
    })
    .Configure(app =>
    {
        app.UseCors(builder => builder.AllowAnyHeader().AllowAnyMethod().SetIsOriginAllowed((host) => true).AllowCredentials());
        app.UseAuthentication();
        app.UseWebSockets();
        app.UseOcelot().Wait();
    })).Build().Run();
```

2- Notify microservice



2- Notify microservice

What it does

- Enable client to stay connected to the stream coming from the signalR server
- Enable ticker service to broadcast coin price updates “ticks” to clients

How it works

- Every client is starting a connection to the SignalR hub
- Client subscribe to the active chart currency and timeframe
- Ticker service connect to the SignalR hub
- Ticker service ask the SignalR hub to publish new tick data to the subscribers those joined this currency and timeframe

NuGet packages

- Microsoft.AspNetCore.SignalR

SignalR definition

ASP.NET SignalR is a library for ASP.NET developers that simplifies the process of adding real-time web functionality to applications.

SignalR handles connection management automatically, and lets you broadcast messages to all connected clients simultaneously, like a chat room. You can also send messages to specific clients. The connection between the client and server is persistent, unlike a classic HTTP connection, which is re-established for each communication.

SignalR uses the new WebSocket transport where available and falls back to older transports where necessary.

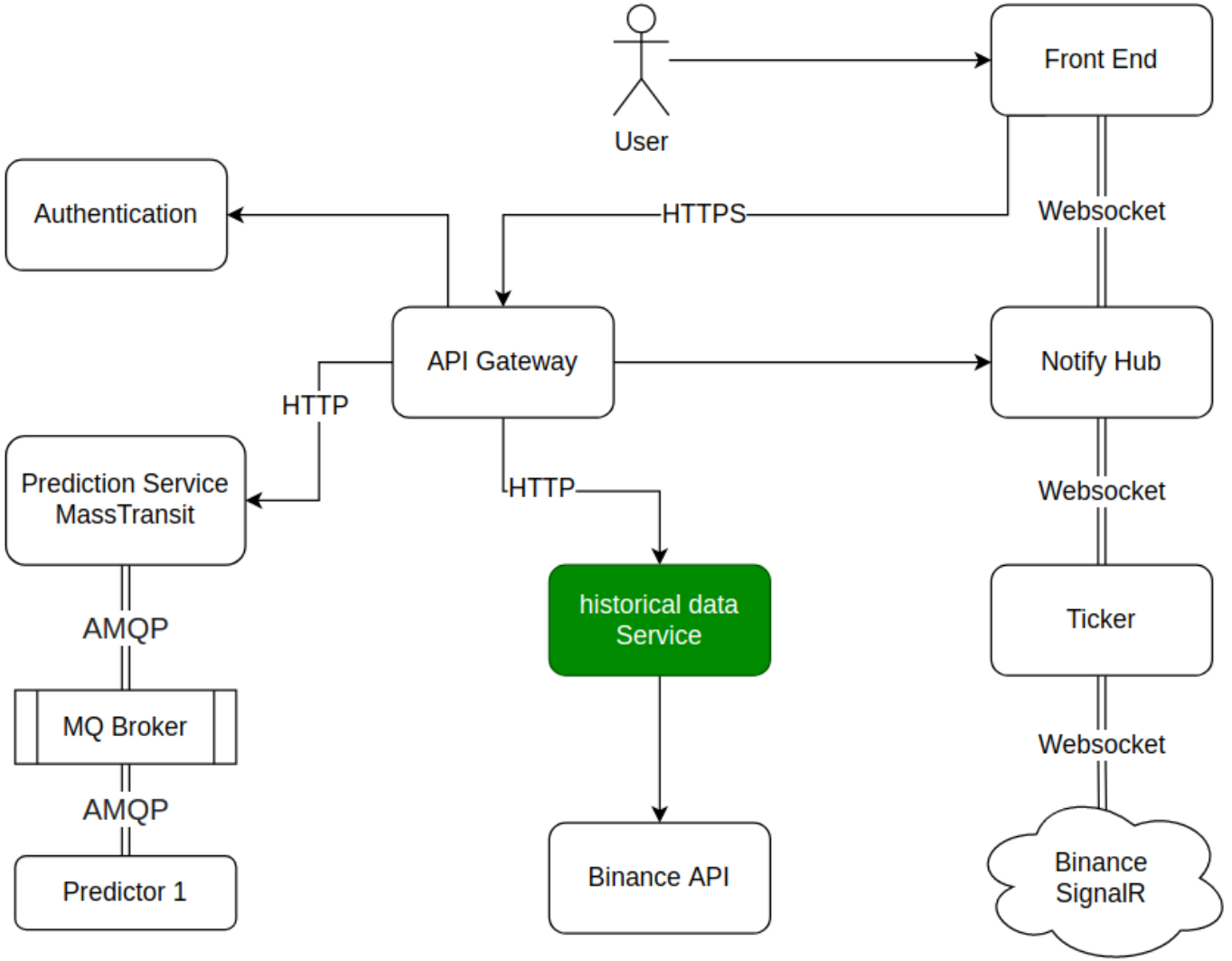
Notify Main

```
Host.CreateDefaultBuilder(args)
    .ConfigureLogging(logging => logging.AddConsole())
    .ConfigureWebHostDefaults(webBuilder => webBuilder.UseUrls("http://0.0.0.0:5100")
    .ConfigureServices(services =>
    {
        services.AddCors();
        services.AddSignalR();
    })
    .Configure(app =>
    {
        app.UseRouting();
        app.UseCors(builder => builder.AllowAnyHeader().AllowAnyMethod().SetIsOriginAllowed((host) => true).AllowCredentials());
        app.UseEndpoints(endpoints => endpoints.MapHub<AppHub>("/ws"));
    })).Build().Run();
```

Notify Hub

```
public class AppHub : Hub
{
    0 references
    public async Task Subscribe(string topic) => await Groups.AddToGroupAsync(Context.ConnectionId, topic);
    0 references
    public async Task Unsubscribe(string topic) => await Groups.RemoveFromGroupAsync(Context.ConnectionId, topic);
    0 references
    public override async Task OnConnectedAsync()
    {
        Console.WriteLine($"{Context.ConnectionId} joined the conversation");
        await base.OnConnectedAsync();
    }
    0 references
    public void Publish(string currency, CandleStick kline)
    {
        Console.WriteLine($"publishing on {currency}");
        Clients.Group(currency).SendAsync(currency, kline);
    }
}
```

3- Historical coin data microservice



3- Historical coin data microservice

What it does

- Expose endpoint to the client to allow the client to load the historical data of a coin

How it works

- It uses a third-party data provider “Binance API” to connect, and retrieve required data

NuGet packages

- BinanceDotNet

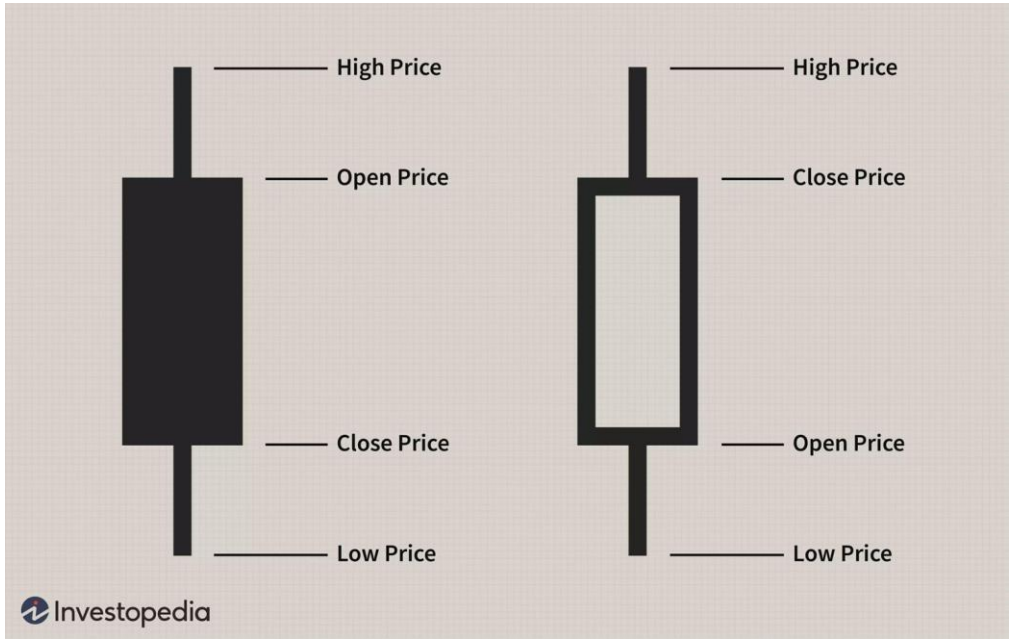
Binance API

Binance is a cryptocurrency exchange that provides a platform for trading various cryptocurrencies. As of April 2021, Binance was the largest cryptocurrency exchange in the world in terms of trading volume.

Binance provide an API to developers to fetch cryptocurrencies data and execute orders

Candlestick definition

A candlestick is a type of price chart used in technical analysis that displays the high, low, open, and closing prices of a security for a specific period.



Historical coin data Main

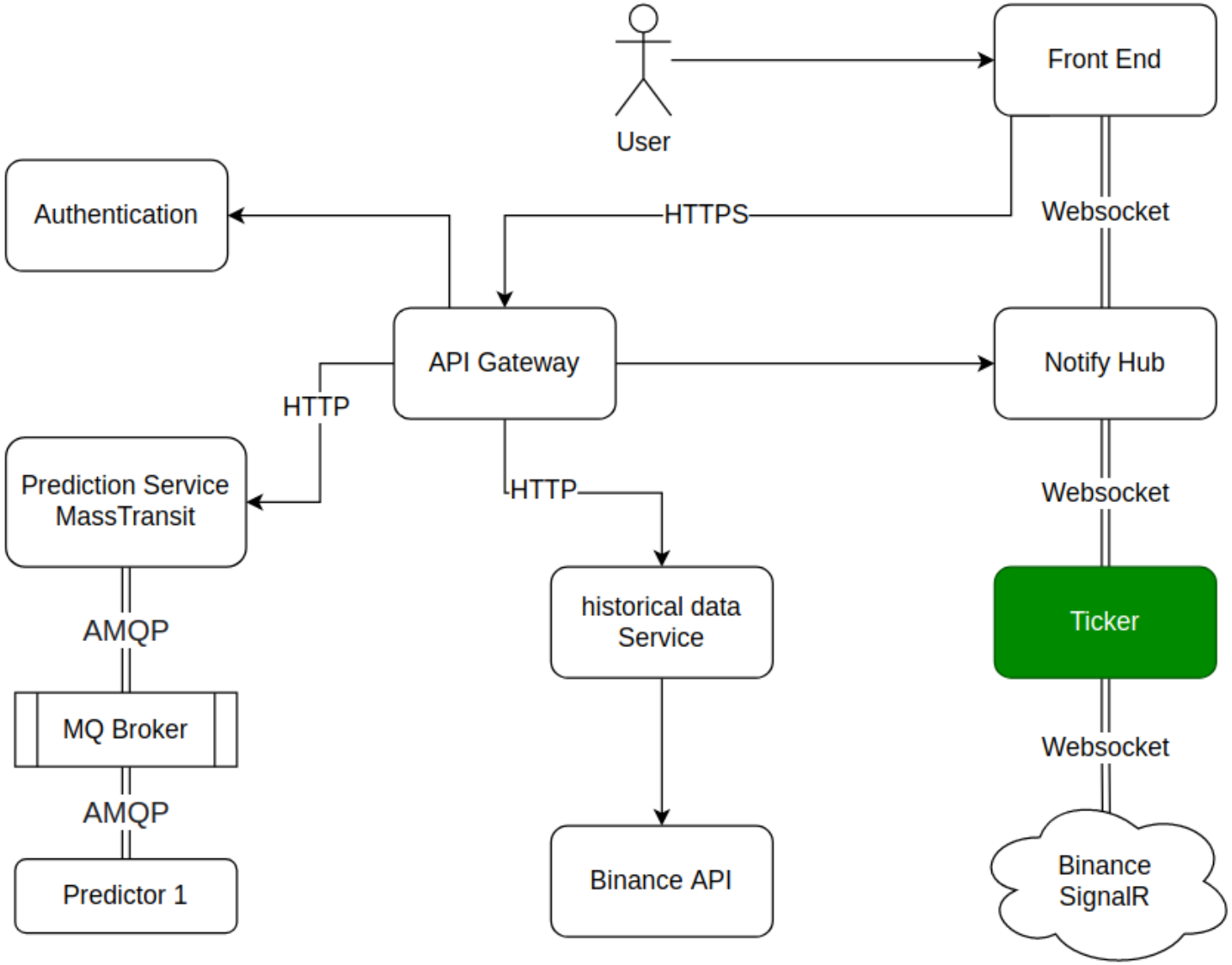
```
string apiKey = "";
string secretKey = "";
Host.CreateDefaultBuilder(args)
    .ConfigureLogging(logging => logging.AddConsole())
    .ConfigureWebHostDefaults(webBuilder => webBuilder.UseUrls("http://0.0.0.0:5200")
    .ConfigureAppConfiguration((hostingContext, config) =>
    { apiKey = config.Build().GetValue<string>("apiKey"); secretKey = config.Build().GetValue<string>("secretKey"); })
    .ConfigureServices(services =>
    {
        var binanceClient = new BinanceClient(new ClientConfiguration() { ApiKey = apiKey, SecretKey = secretKey });
        services.AddResponseCompression();
        services.AddApiVersioning(config =>
        { config.DefaultApiVersion = new ApiVersion(1, 0); config.AssumeDefaultVersionWhenUnspecified = true; });
        services.AddSingleton<IBinanceClient>(binanceClient);
        services.AddCors();
        services.AddResponseCaching();
        services.AddControllers();
    })
    .Configure(app =>
    {
        app.UseRouting();
        app.UseResponseCompression();
        app.UseResponseCaching();
        app.UseCors(builder => builder.AllowAnyHeader().AllowAnyMethod().SetIsOriginAllowed((host) => true).AllowCredentials());
        app.UseEndpoints(endpoints => endpoints.MapControllers());
    })).Build().Run();
```

Historical coin data Controller

```
public class DataController : ControllerBase
{
    //BTCUSDT?tframe=4&limit=H1&startDate=10000&endDate=1619945168
    [ResponseCache(Duration = 50, Location = ResponseCacheLocation.Any, NoStore = false)]
    [HttpGet("{currency}")]
    0 references
    public async Task<IEnumerable<CandleStick>> Get([FromServices] IBinanceClient binanceClient, string currency,
        int? limit, long? startDate, long? endDate, string tframe)
    {
        KlineInterval interval = tframe switch
        {
            "M1" => KlineInterval.OneMinute,
            "M5" => KlineInterval.FiveMinutes,
            "M15" => KlineInterval.FifteenMinutes,
            "M30" => KlineInterval.ThirtyMinutes,
            "H1" => KlineInterval.OneHour,
            "H4" => KlineInterval.FourHours,
            "D1" => KlineInterval.OneDay,
            "W1" => KlineInterval.OneWeek,
            _ => KlineInterval.OneHour
        };
        var opts = new GetKlinesCandlesticksRequest { Interval = interval, Limit = limit, Symbol = currency };
        opts.StartTime = (startDate.HasValue) ? DateTimeOffset.FromUnixTimeSeconds(startDate.Value).DateTime.ToLocalTime() : null;
        opts.EndTime = (endDate.HasValue) ? DateTimeOffset.FromUnixTimeSeconds(endDate.Value).DateTime.ToLocalTime() : null;

        var data = await binanceClient.GetKlinesCandlesticks(opts);
        var result = data.Select(d => new CandleStick()
        {
            Open = d.Open,
            Close = d.Close,
            High = d.High,
            Low = d.Low,
            Volume = d.Volume,
            Timestamp = d.OpenTime.ToToUnixTimestamp()
        }).ToList();
        return result;
    }
}
```

4- Ticker microservice



4- Ticker microservice

What it does

- Publish all candlestick changes of all coins on all timeframes

How it works

- Connect to third party stream provider “Binance hub”
- Subscribe to all coins and all timeframes
- Connect to our application SignalR hub
- Republish all messages received from Binance hub to our application SignalR hub

NuGet packages

- Microsoft.AspNetCore.SignalR.Client
- Microsoft.Extensions.Configuration

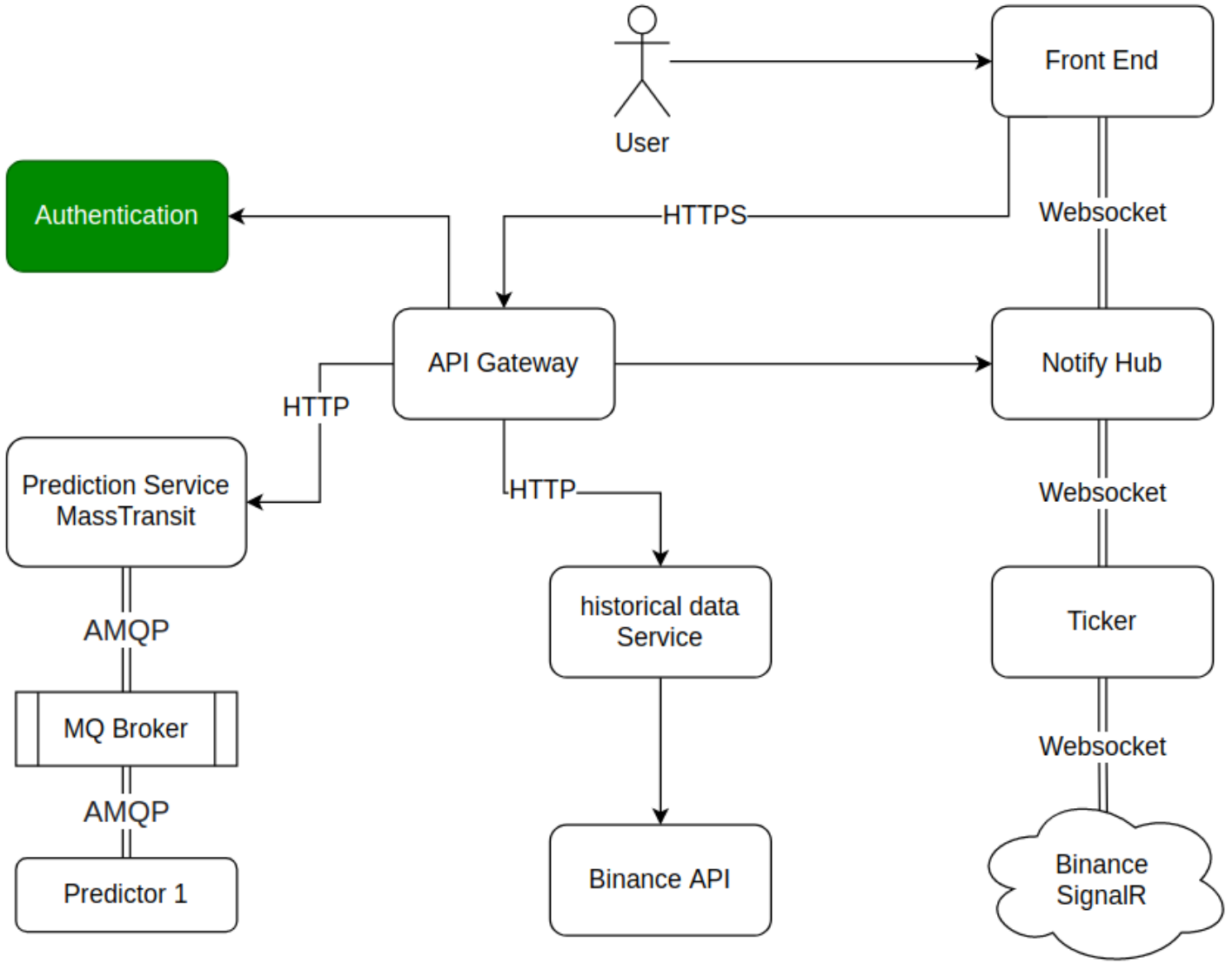
Ticker Main

```
var config = new ConfigurationBuilder().AddJsonFile($"appsettings.json", true, true).AddEnvironmentVariables().Build();
string apiKey = config.GetValue<string>("apiKey");
string secretKey = secretKey = config.GetValue<string>("secretKey");
var SignalRConnection = new HubConnectionBuilder().WithUrl("http://localhost:5100/ws",
    ops => ops.AccessTokenProvider = () => Task.FromResult("Ticker")).Build();
SignalRConnection.Closed += async (error) =>{ await Task.Delay(new Random().Next(0, 5) * 1000);  await SignalRConnection.StartAsync();};
await SignalRConnection.StartAsync().ContinueWith(t => Console.WriteLine("Connection Started"));
var binanceClient = new BinanceClient(new ClientConfiguration { ApiKey = apiKey, SecretKey = secretKey });
var binanceWebSocketClient = new DisposableBinanceWebSocketClient(binanceClient);

var currencies = new List<string> { "BTCUSDT", "ETHUSDT" };
var timeframe = new Dictionary<string, KlineInterval> { {"M1", KlineInterval.OneMinute } , {"M5", KlineInterval.FiveMinutes },
    {"M15", KlineInterval.FifteenMinutes },{"M30", KlineInterval.ThirtyMinutes }, {"H1", KlineInterval.OneHour },
    {"H4", KlineInterval.FourHours },{"D1", KlineInterval.OneDay } };

currencies.ForEach(c =>
timeframe.ToList().ForEach(t =>
    binanceWebSocketClient.ConnectToKlineWebSocket(c, t.Value, data =>
    {
        var klineObj = new CandleStick()
        {
            Close = data.Kline.Close,
            High = data.Kline.High,
            Low = data.Kline.Low,
            Open = data.Kline.Open,
            Volume = data.Kline.Volume,
            Timestamp = data.Kline.StartTime.ToToUnixTimestamp()
        };
        SignalRConnection.InvokeAsync("Publish", $"{c}/{t.Key}", klineObj);
    })
));
new ManualResetEvent(false).WaitOne();
```

5- Authentication microservice



What it does

- Expose endpoint to allow client to login using username and password and get token

How it works

- Client post username and password to login endpoint
- Microservice validate the username and password are correct
  - If correct microservice generate a new token and return it
  - if incorrect it return unauthorized HTTP status.

NuGet packages

- Microsoft.AspNetCore.Authentication.JwtBearer

Authentication Main

```
string stringKey = "";
Host.CreateDefaultBuilder(args)
    .ConfigureLogging(logging => logging.AddConsole())
    .ConfigureWebHostDefaults(webBuilder => webBuilder.UseUrls("http://0.0.0.0:5300")
    .ConfigureAppConfiguration((hostingContext, config) => stringKey = config.Build().GetValue<string>("Secret"))
    .ConfigureServices(services =>
    {
        var key = Encoding.UTF8.GetBytes(stringKey);
        services.AddSingleton<JWTAuthenticationManager>(new JWTAuthenticationManager(key));
        services.AddCors();
        services.AddControllers();
    })
    .Configure(app =>
    {
        app.UseRouting();
        app.UseCors(builder => builder.AllowAnyHeader().AllowAnyMethod().SetIsOriginAllowed((host) => true).AllowCredentials());
        app.UseEndpoints(endpoints => endpoints.MapControllers());
    })).Build().Run();
```

Authentication Manager

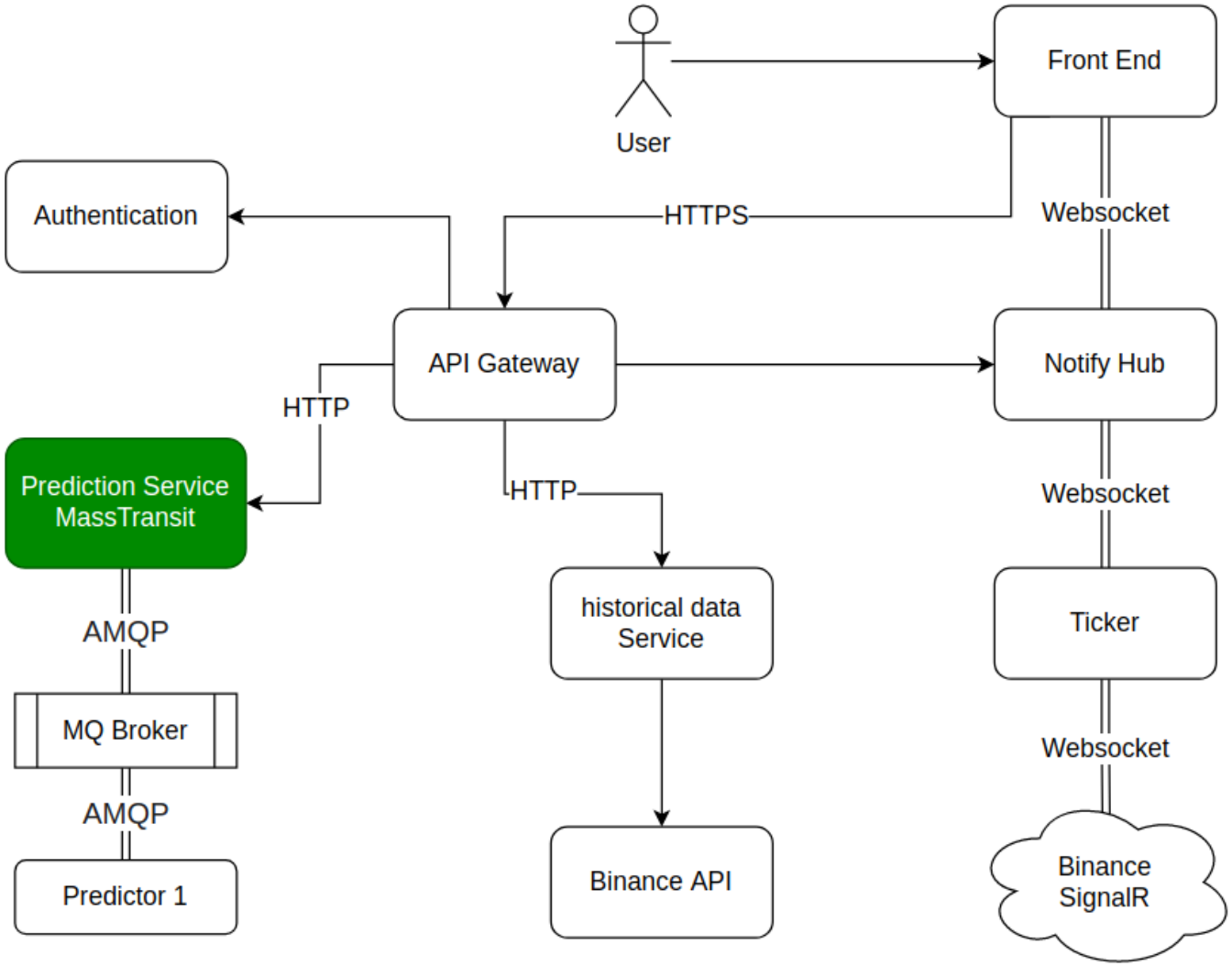
```
public class JWTAuthenticationManager
{
    readonly IDictionary<string, string> users = new Dictionary<string, string> {{ "demo", "demo" },{ "admin", "password" }};
    private readonly byte[] tokenKey;
    1 reference
    public JWTAuthenticationManager(byte[] tokenKey) => this.tokenKey = tokenKey;
    public record TokenResponse(string access_token, long expires);
    1 reference
    public TokenResponse Authenticate(string username, string password)
    {
        if (!users.Any(u => u.Key == username && u.Value == password)) return null;
        var expires = DateTime.UtcNow.AddMinutes(1);
        var tokenHandler = new JwtSecurityTokenHandler();
        var tokenDescriptor = new SecurityTokenDescriptor
        {
            Subject = new ClaimsIdentity(new Claim[] {new Claim(ClaimTypes.Name, username) }),
            Expires = expires,
            SigningCredentials = new SigningCredentials(new SymmetricSecurityKey(tokenKey), SecurityAlgorithms.HmacSha256Signature)
        };
        var desc = tokenHandler.CreateToken(tokenDescriptor);
        var token = new TokenResponse(tokenHandler.WriteToken(desc), expires.ToToUnixTimestamp());
        return token;
    }
}
```

Authentication Controller

```
public class AuthController : ControllerBase
{
    public record GetTokenRequest(string Username, string Password);

    [AllowAnonymous]
    [HttpPost("access_token")]
    0 references
    public IActionResult Authorize([FromServices] JWTAuthenticationManager authenticationManager, [FromBody] GetTokenRequest request)
    {
        var token = authenticationManager.Authenticate(request.Username, request.Password);
        return token == null ? Unauthorized() : Ok(token);
    }
}
```

6- Prediction microservice



6- Prediction microservice

What it does

- Expose endpoint to allow the client to ask for a prediction of specific currency on a timeframe

How it works

- Client send a GET request asking for prediction of currency x on timeframe z
- Microservice prepare a *PredictionRequest* and publish it into its RabbitMQ service bus
- Microservice wait to get its response on its request
- Microservice fetch *PredictionResult* and return it to the client

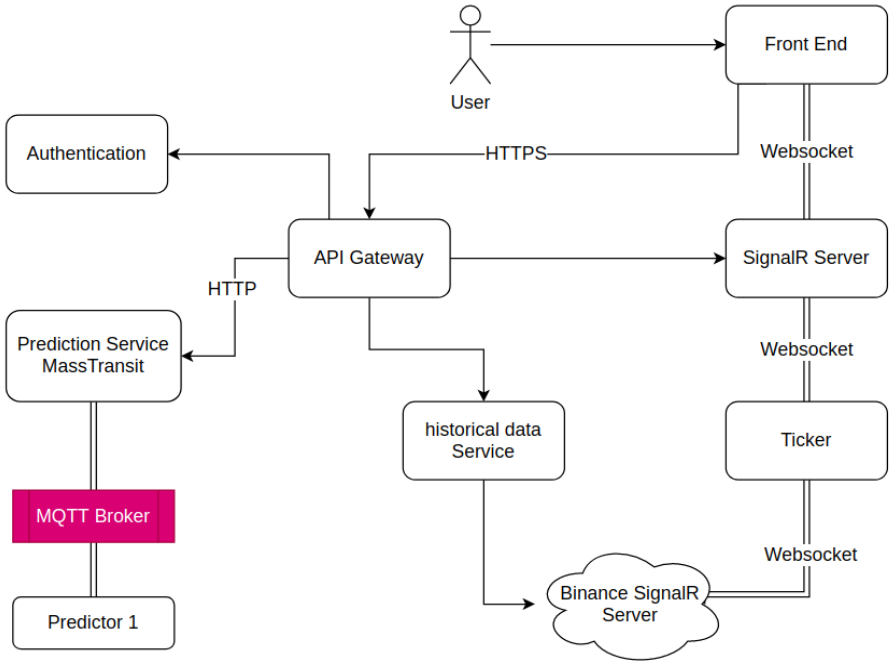
NuGet packages

- MassTransit
- MassTransit.AspNetCore
- MassTransit.RabbitMQ
- MassTransit.Extensions.DependencyInjection

RabbitMQ

RabbitMQ is the most widely deployed open-source message broker.

RabbitMQ is originally implemented the Advanced Message Queuing Protocol (AMQP) and has since been extended with a plug-in architecture to support Streaming Text Oriented Messaging Protocol (STOMP), MQ Telemetry Transport (MQTT), and other protocols.



MassTransit

MassTransit is free software/open-source .NET-based Enterprise Service Bus (ESB) software that helps .NET developers route messages over RabbitMQ, Azure Service Bus, SQS, and ActiveMQ service busses.

It supports multicast, versioning, encryption, sagas, retries, transactions, distributed systems and other features.

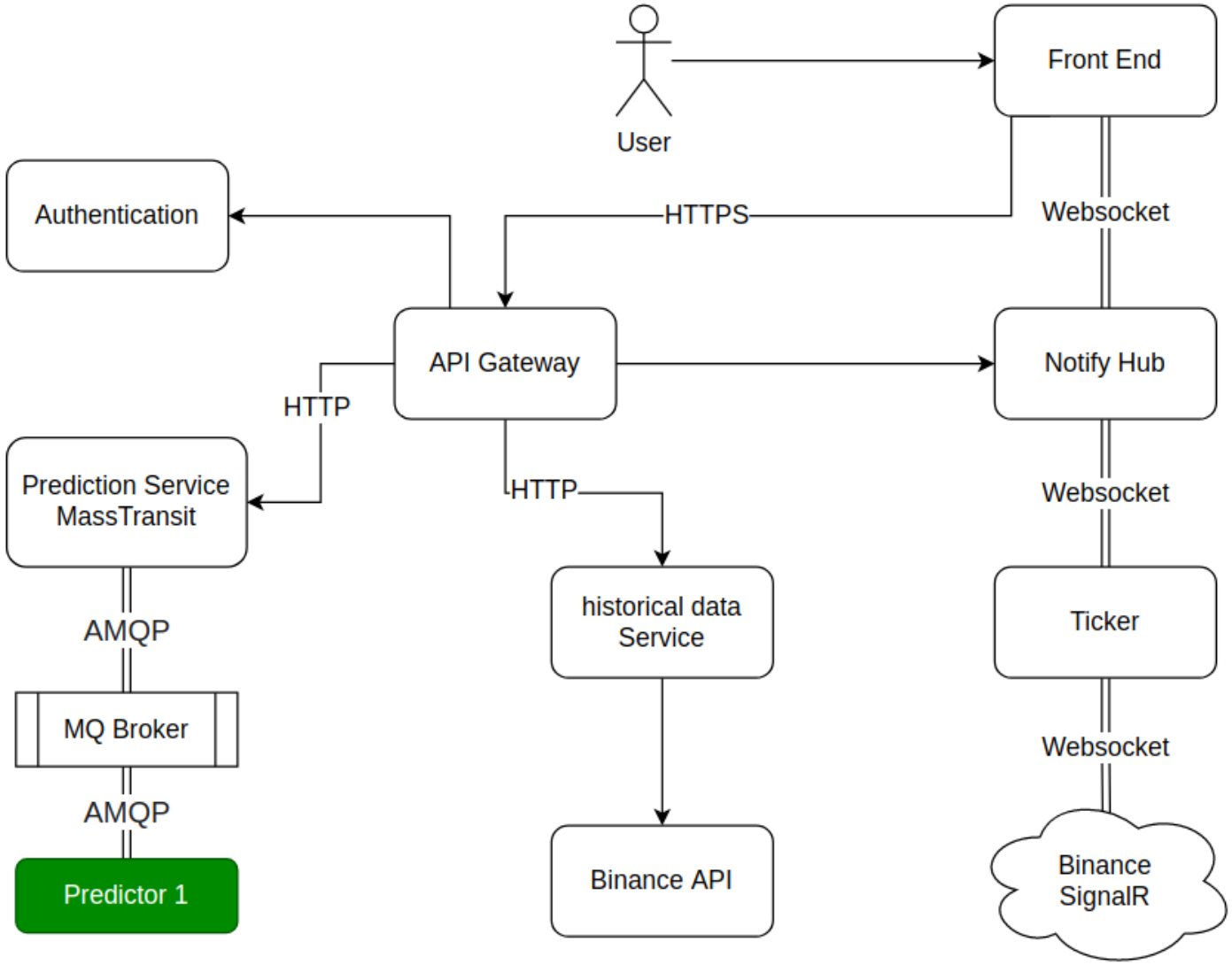
Prediction Main

```
Host.CreateDefaultBuilder(args)
    .ConfigureLogging(logging => logging.AddConsole())
    .ConfigureWebHostDefaults(webBuilder => webBuilder.UseUrls("http://0.0.0.0:5500")
    .ConfigureServices(services =>
    {
        services.AddControllers();
        services.AddMassTransitHostedService().AddMassTransit(x =>
        {
            x.AddRequestClient<PredictionRequest>();
            x.AddBus(context => Bus.Factory.CreateUsingRabbitMq(c =>
            {
                c.Host("rabbitmq://localhost");
                c.ConfigureEndpoints(context);
            }));
        });
    })
    .Configure(app =>
    {
        app.UseRouting();
        app.UseCors(builder => builder.AllowAnyHeader().AllowAnyMethod().SetIsOriginAllowed((host) => true).AllowCredentials());
        app.UseEndpoints(endpoints => endpoints.MapControllers());
    })).Build().Run();
```

Prediction Controller

```
public class PredictController : ControllerBase
{
    [HttpGet("{currency}")]
    0 references
    public async Task<ActionResult<PredictionResult>> Get
        ([FromServices] IRequestClient<PredictionRequest> client, string currency, int? limit, long? endDate, string tframe)
    {
        var payload = new PredictionRequest(currency, limit, endDate, tframe);
        var request = client.Create(payload);
        var response = await request.GetResponse<PredictionResult>();
        return Ok(response);
    }
}
```

7- Time Series Forecaster microservice



7- Time Series Forecaster microservice

What it does

- Predicate upcoming 20 candlestick close prices for a currency on a time frame

How it works

- Microservice is active listening into RabbitMQ queue waiting for any PredictionRequest
- When there is a new PredictionRequest , Microservice fetch PredictionRequest and then
  - Fetch historical data from historical data microservice for request's currency and time frame, etc.
  - Train a machine learning model using a predefined training pipeline with the data has fetched.
  - Generate a time series forecasting engine for this trained model and forecast the upcoming prices.
  - Publish PredictionResult containing the forecast result, which is going to picked up by Prediction service.

NuGet packages

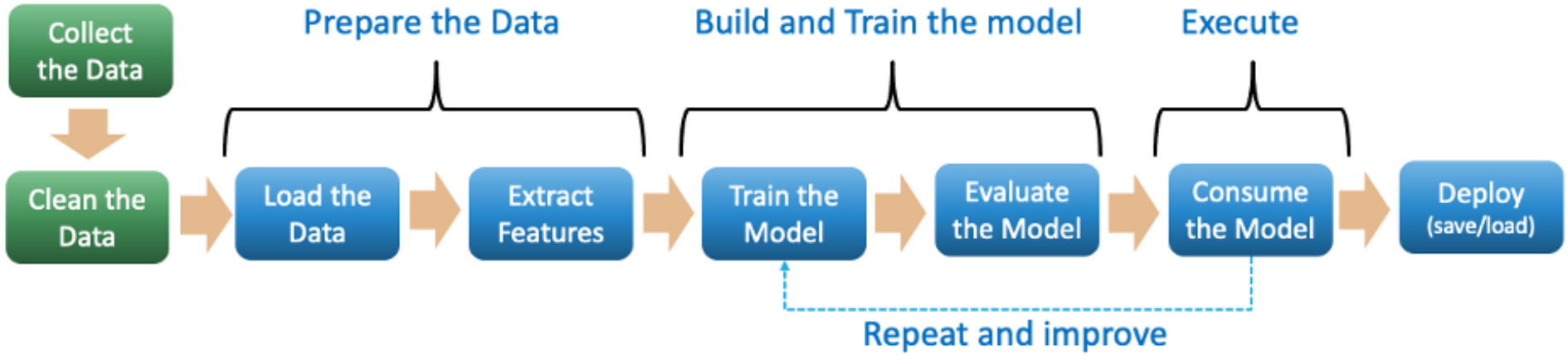
- MassTransit
- MassTransit.RabbitMQ
- Microsoft.ML
- Microsoft.ML.TimeSeries

ML.NET

ML.NET is a machine learning framework for .NET developers, you can use ML.NET to integrate custom machine learning models into your .NET applications.

You can use ML.NET for many scenarios, such as sentiment analysis, price prediction, product recommendation, sales forecasting, image classification, object detection, and more!.

Building an ML Model involves few steps



Time Series Forecaster Main

```
Console.WriteLine("Predictor1 started");
var busControl = Bus.Factory.CreateUsingRabbitMq(cfg => cfg.ReceiveEndpoint("prediction-requests", e => e.Consumer<PredictionRequestConsumer>()));
await busControl.StartAsync();
new ManualResetEvent(false).WaitOne();
```

PredictionRequestConsumer

```
1 reference
class PredictionRequestConsumer : IConsumer<PredictionRequest>
{
    0 references
    public async Task Consume(ConsumeContext<PredictionRequest> context)
    {
        Console.WriteLine($"Value: {context.Message.currency} - {context.Message.tframe}");
        var data = await FetchData(context.Message);
        PredictionResult result = Forcast(data);
        await context.RespondAsync(result);
    }
```

FetchData

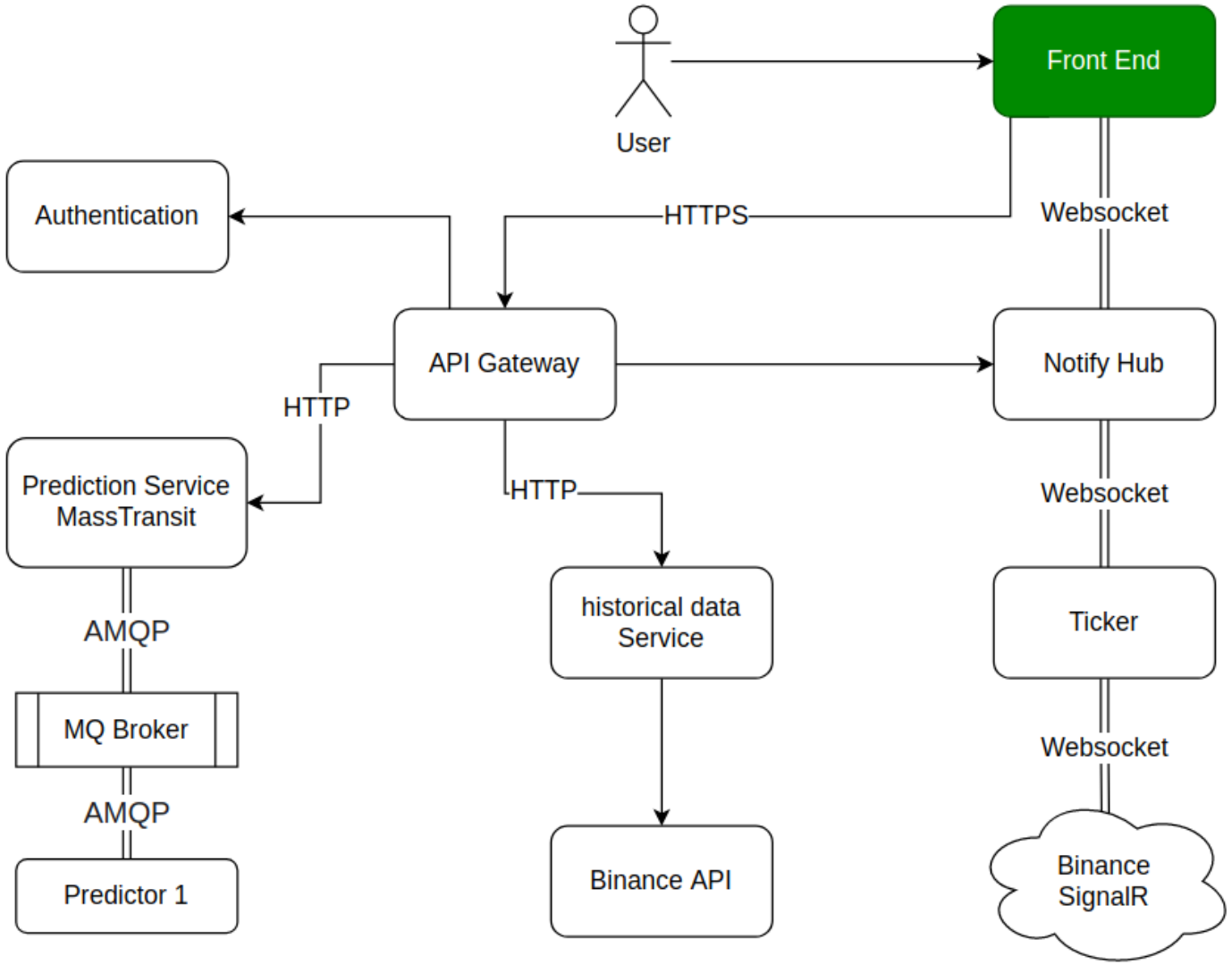
```
static readonly HttpClient client = new();
public record InputModel(float Close);
static string host = "http://localhost:5200";
1 reference
static async Task<IEnumerable<InputModel>> FetchData(PredictionRequest request)
{
    try
    {
        var response = await
            client.GetAsync($"{host}/data/{request.currency}?tframe={request.tframe}&limit={request.limit}&endDate={request.endDate}");
        response.EnsureSuccessStatusCode();
        var data = await response.Content.ReadFromJsonAsync<List<CandleStick>>();
        var converted = data.Select(d => new InputModel(decimal.ToSingle(d.Close)));
        return converted;
    }
    catch (HttpRequestException e)
    {
        Console.WriteLine("\nException Caught!");
        Console.WriteLine("Message :{0} ", e.Message);
        return default;
    }
}
```

Forecast

```
static PredictionResult Forcast(IEnumerable<InputModel> data)
{
    int series = 7;
    int window = 2;
    int horizon = 20;
    float confidenceLevel = 0.75f;
    if (data == default) return default;
    var context = new MLContext();
    var dataview = context.Data.LoadFromEnumerable(data);

    var pipeline = context.Forecasting.ForecastBySsa(
        nameof(PredictionResult.Value),
        nameof(InputModel.Close),
        windowSize: window,
        seriesLength: series,
        trainSize: data.Count(),
        horizon: horizon,
        confidenceLevel: confidenceLevel,
        confidenceLowerBoundColumn: nameof(PredictionResult.Low),
        confidenceUpperBoundColumn: nameof(PredictionResult.High)
        );
    var model = pipeline.Fit(dataview);
    var forecastingEngine = model.CreateTimeSeriesEngine<InputModel, PredictionResult>(context);
    var forecasts = forecastingEngine.Predict();
    return forecasts;
}
```

8- Frontend microservice



8- Frontend microservice

What it does

- Login / authenticate and store access token and mange regenerate/renew token when expires
- Fetch historical data for active selected currency and timeframe
- Render historical data into a candlestick chart
- Enable list of popular indicators (chart indicators and external panel indicators)
- Load forecasting for active selected currency and timeframe
- Enable a custom chart indicator to show forecasting area
- Connect to Notify server (SignalR Hub)
- Subscribe to SignalR topic “{selected_currency}/{selected_timeframe}”
- Handle subscription messages with new candlestick changes and render changes into the active chart

How it works

- Vue Single Page Application (includes routing capabilities “vue router”)
- Single SignalR connection shared between all the pages and managed by subscriptions

Vue

Vue is a progressive framework for building user interfaces.

The advantages

- Lightweight
- Virtual DOM performance and rendering
- Great documentation and community support
- Easy to get started
- Readability and single-file components
- And More!

Tailwindcss

Tailwindcss is a utility-first CSS framework packed with classes like flex, pt-4, text-center and rotate-90 that can be composed to build any design, directly in your markup.

Rapidly build modern websites without ever leaving your HTML,
No more you need to have style.css file 😀

The advantages

- It’s tiny in production.
- Responsive everything.
- Build whatever you want, seriously.
- Extend it, tweak it, change it.

AppClient::GetAccessToken

```
async GetAccessToken() {
    if (this.expires != null && Date.now() < this.expires) return;
    console.log("GetAccessToken")
    const response = await fetch(
        `https://${this.host}:5000/api/auth/access_token`, {
            method: "POST",
            headers: {
                Accept: "application/json",
                "Content-Type": "application/json",
            },
            body: JSON.stringify({
                Username: this.username,
                Password: this.password
            }),
        }
    );
    const data = await response.json();
    this.token = data.access_token;
    this.expires = data.expires;
}
```

AppClient::LoadData & LoadForecast

```
async LoadData(currency, tframe, end, limit) {
    await this.GetAccessToken();
    console.log("LoadData")
    const response = await fetch(`https://${this.host}:5000/api/data/${currency}?tframe=${tframe}&limit=${limit}&endDate=${end}`, {
        method: "GET",
        headers: new Headers({
            Authorization: "Bearer " + this.token,
        }),
    })
    const json = await response.json();
    return json;
}

async LoadForcast(currency, tframe, end, limit) {
    await this.GetAccessToken();
    console.log("LoadForcast")
    const response = await fetch(`https://${this.host}:5000/api/forcast/${currency}?tframe=${tframe}&limit=${limit}&endDate=${end}`, {
        method: "GET",
        headers: new Headers({
            Authorization: "Bearer " + this.token,
        }),
    })
    const json = await response.json();
    return json;
}
```

AppClient::SignalRClient

```
async Connect() {
    if (this.connection != null && this.connection.state == "Connected") return;
    console.log("New connection for SignalR")
    this.connection = new signalR.HubConnectionBuilder()
        .configureLogging(signalR.LogLevel.Debug)
        .withUrl(`https://${this.host}:5000/ws`, {
            accessTokenFactory: () => this.token,
        })
        .withAutomaticReconnect([0, 0, 10000])
        .build();
    this.connection.onreconnecting((error) =>
        console.log(`Connection lost due to error "${error}". Reconnecting.`)
    );
    await this.connection.start();
    window.connection = this.connection;
}
async Subscribe(currency, tframe, DataChange) {
    await this.Connect();
    let topic = `${currency}/${tframe}`
    this.connection.on(topic, DataChange);
    this.connection.invoke("Subscribe", topic).catch(console.log);
    console.log(`Subscribed to ${topic}`)
}
async Unubscribe(currency, tframe) {
    if (this.connection == null || this.connection.state != "Connected") return;
    let topic = `${currency}/${tframe}`
    this.connection.invoke("Unsubscribe", topic).catch(console.log);
    console.log(`Unsubscribed to ${topic}`)
}
```

Candlestick chart page JavaScript

```
export default {
  name: "Chart",
  props: ["api"],
  data() {…
  },
  async mounted() {
    this.InitChart();
    let start = parseInt(Date.now()/1000);
    let data = await this.api.client.LoadData(this.currency, this.tframe, start, this.numberOfBars );
    this.LoadChartData(data);
    forecast_data = await this.api.client.LoadForcast( this.currency,this.tframe,start, this.numberOfBars );
    this.forecast_ready = true;
    console.log(forecast_data);
    await this.api.client.Subscribe(this.currency, this.tframe, this.DataChange);
  },
  methods: {
    InitChart: function () {…
    },
    LoadChartData: function (data) {…
    },
    DataChange: function (data) {…
    },
    setCandleTechnicalIndicator: function (type) {…
    },
    setSubTechnicalIndicator: function (type) {…
    },
    showForecast: function () {…
    },
  },
  unmounted() {
    this.api.client.Unubscribe(this.currency, this.tframe);
  },
};
</script>
```

Lessons learned

- .NET 5 helps you to develop microservices easily.
- Binance and many other cryptocurrencies broker provide public API to get data feeds.
- Masstransit is break through into microservices for .NET developers.
- ML.NET support building many machine learning applications with less code.
- Vue.js framework helps you to build complete front end application with less code than other frameworks.
- SignalR binds both client and server together and allow remote method invocation to both.


THANK
YOU !