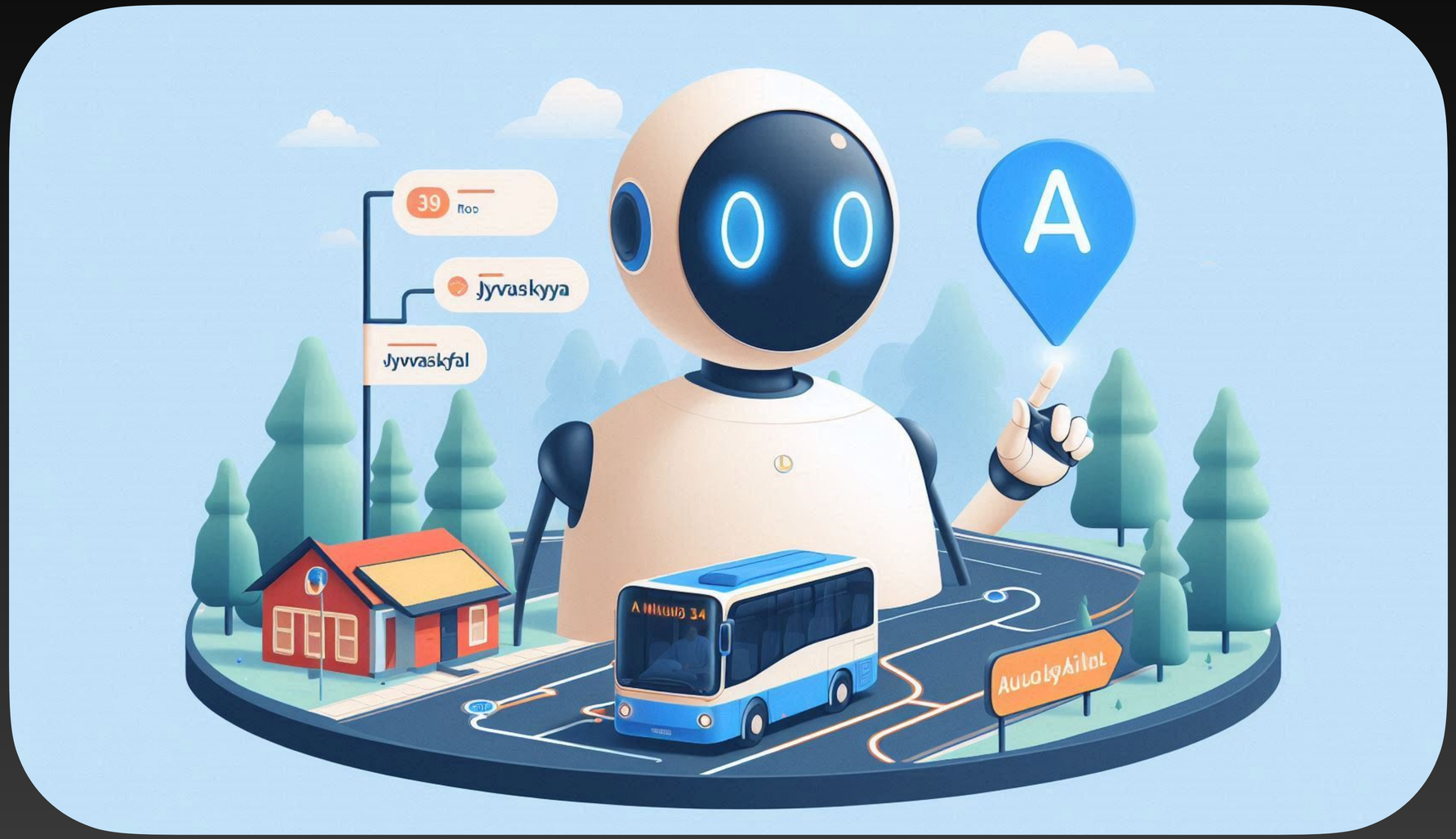
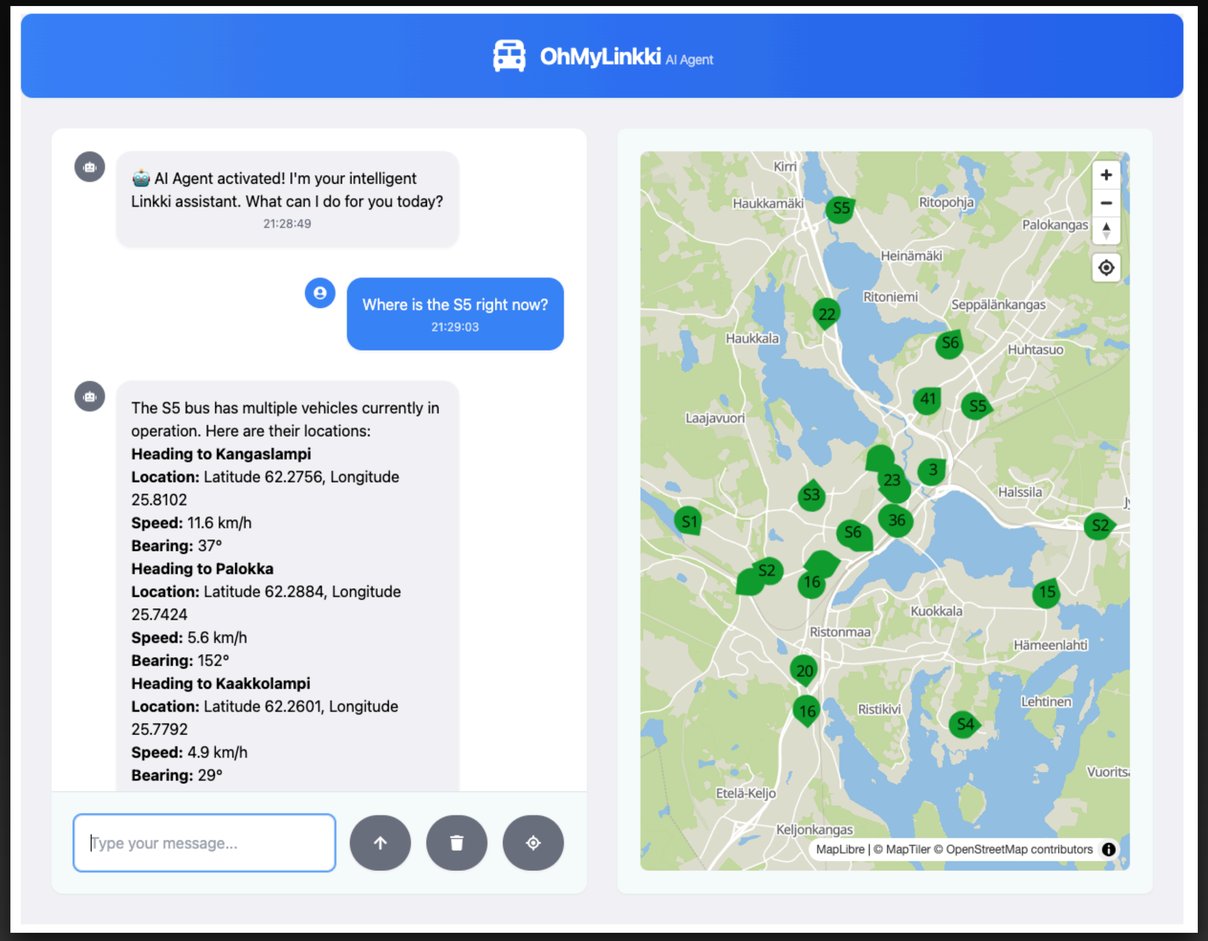
OhMyLinkki: Azure AI-Powered Real-Time Bus Tracking Assistant

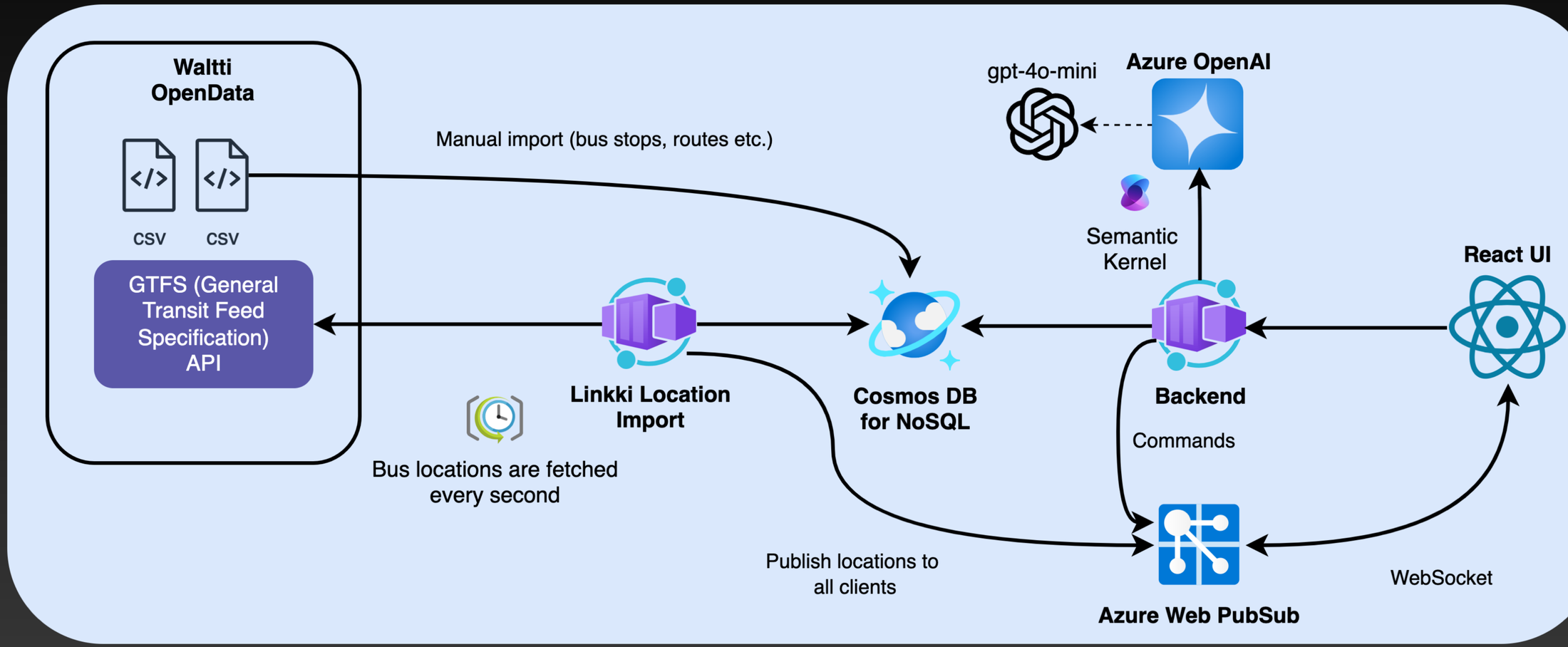


OhMyLinkki

- **Linkki** is the public transport in Jyväskylä Region
- **OhMyLinkki**
  - Real-time bus tracking: Show Linkki buses on the map in real time
  - Interactive Chat Interface: Users can ask about bus information, including locations, bus stops, and schedules.
  - It is designed to explore and experiment with AI technologies



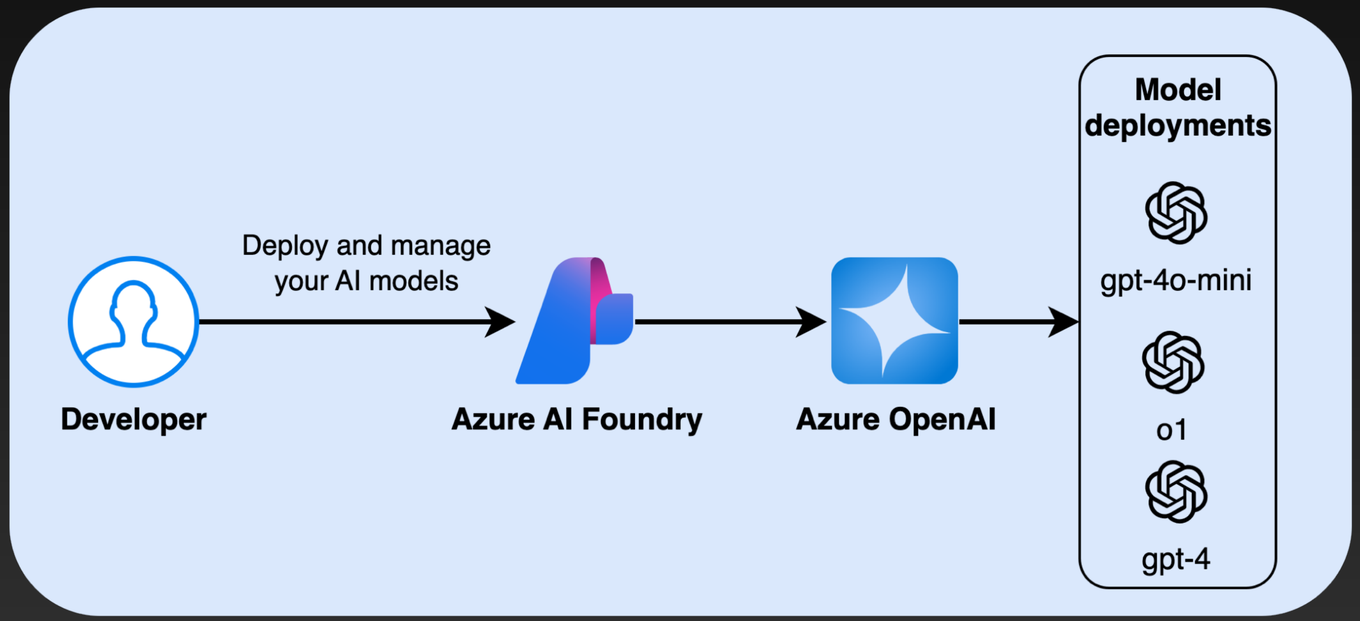
OhMyLinkki
Architecture



Key Components and Services

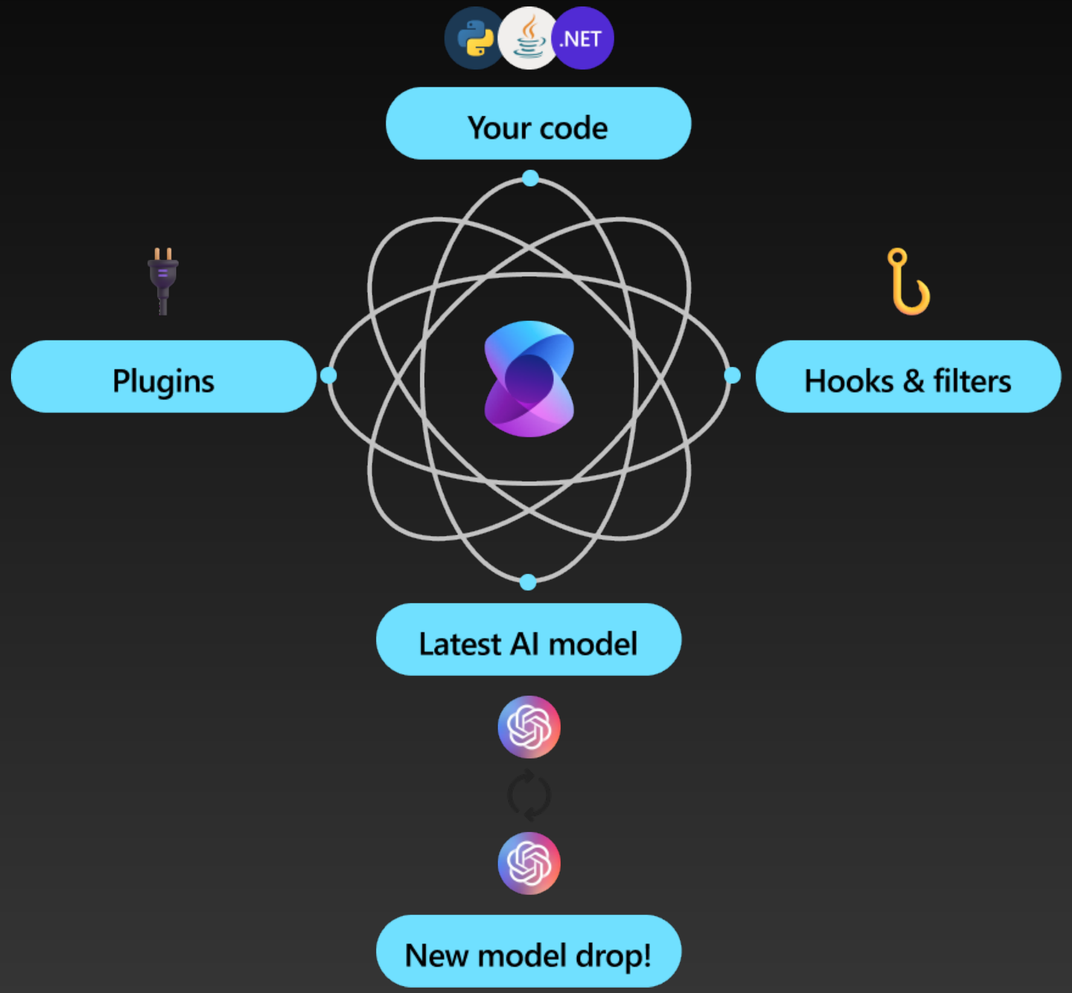
Azure OpenAI Service

- Azure managed service to deploy, run and manage OpenAI models
  - Supports model like GPT-4, GPT-4o, o1 and other OpenAI models.
  - Allows fine tuning of models
- Models can be deployed from Azure AI Foundry or with IaC (ARM/Bicep)
- Provides a secure endpoint to access your models
    - Role-based access control (Azure RBAC)
- Seamlessly integrates with other Azure Services
    - Semantic Kernel
    - Logic App
    - Azure Functions



Semantic Kernel

- Open source Orchestration library/framework/tool that helps creating AI-powered applications and AI agents
  - Created by Microsoft
  - Supports C#, Python and Java
- Support various AI services using connector pattern
  - OpenAI, AzureOpenAI, Ollama, HugginFace
- With plugins, you can enable AI to extend its knowledge
  - AI models can call your functions and access your data in external services such databases or APIs
- Use filters to audit function calls or validating the prompts
  - Implement logging and monitoring
  - Authorisation: is user allowed to use this plugin function

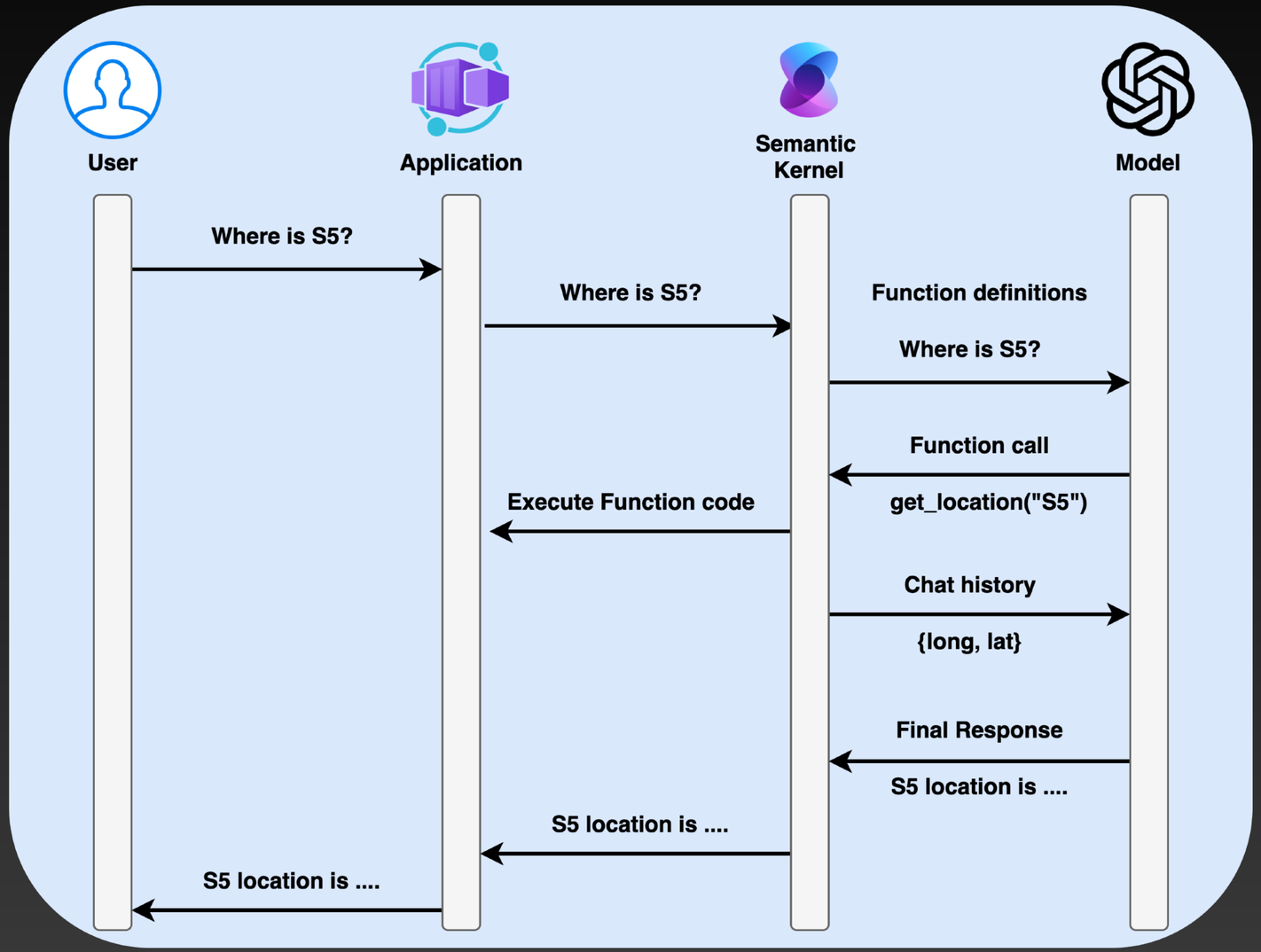


Source: https://learn.microsoft.com/en-us/semantic-kernel/overview/

What are Plugins?

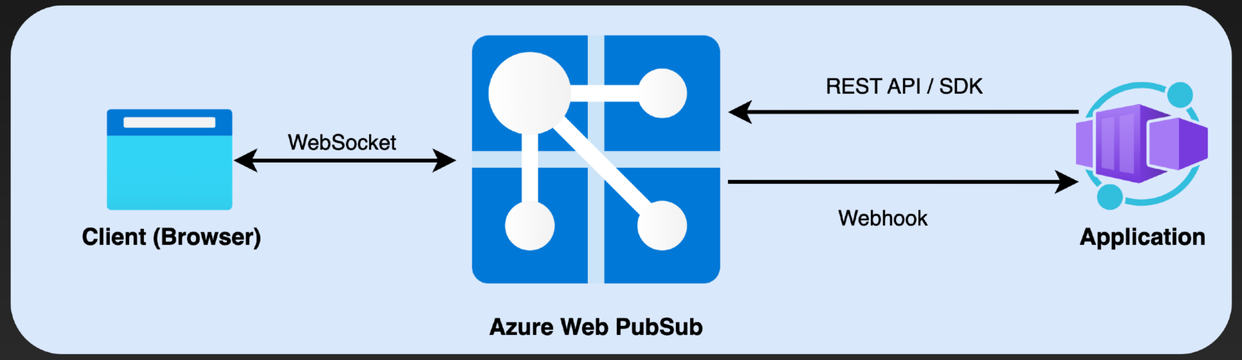
- Plugins are group of functions registered in Semantic Kernel
- Similar concepts are called tools or actions in other platforms
- Semantic Kernel leverages the function calling capabilities of AI models
- The model decides on its own which functions to call and when

  ✳ Agent



Azure Web PubSub

- Azure Managed service to build web applications with real-time features
- Supports standard WebSockets
- Web PubSub service takes care all of WebSocket communication
  - You can focus on building features in your application, not on managing WebSocket connections
- Easy to use, no need for external libraries on the client side (browser).
- Example use cases
  - AI applications
  - Location tracking
  - Chat applications
  - Live dashboards & monitoring



Show me the code!

Semantic Kernel - Agent Setup

```
builder.Services.AddSingleton<LinkkiPlugin>();
builder.Services.AddTransient((sp) =>
{
    KernelPluginCollection pluginCollection = [];
    pluginCollection.AddFromObject(sp.GetRequiredService<LinkkiPlugin>());
    return new Kernel(sp, pluginCollection);
});
```

```
builder.Services.AddSingleton<IChatCompletionService>(sp =>
{
    var options = sp.GetRequiredService<IOptions<OpenAiOptions>>().Value;
    return new AzureOpenAIChatCompletionService(options.DeploymentName, options.Endpoint,
azureCredential,
        options.DeploymentName);
});
```

```
builder.Services.AddTransient(sp =>
{
    var kernel = sp.GetRequiredService<Kernel>();
    var options = sp.GetRequiredService<IOptions<OpenAiOptions>>().Value;
    ChatCompletionAgent agent =
        new()
        {
            Name = AgentInstructions.LinkkiAgentName,
            Instructions = AgentInstructions.LinkkiAgentInstructions,
            Kernel = kernel,
            Arguments = new KernelArguments(new AzureOpenAIPromptExecutionSettings()
            {
                ToolCallBehavior = ToolCallBehavior.AutoInvokeKernelFunctions,
                Temperature = options.Temperature
            })
        };
    return agent;
});
```

Semantic Kernel - Agent Instructions

```
public const string LinkkiAgentName = "LinkkiAgent";

public const string LinkkiAgentInstructions =
    """
    You are a friendly and knowledgeable AI Agent dedicated to helping users
    with Linkki bus lines in Jyväskylä.

    Here are some guidelines for your responses:

    Answer questions about bus lines and locations.
    Provide real-time information about the current location of buses.
    All your responses should be in Markdown format.
    Keep answers short and to the point.
    Bearing needs to be in compass directions.

    """;
```

Semantic Kernel - Chat API

```
app.MapPost("api/chat-agent",
    async (UserChatMessage userMessage, ChatCompletionAgent agent, IChatHistoryProvider
chatHistoryProvider,
        IHttpContextAccessor httpContextAccessor, IOptions<OpenAiOptions> openAiOptions) =>
    {
        if (httpContextAccessor.HttpContext != null)
        {
            httpContextAccessor.HttpContext.Items["userId"] = userMessage.UserId;
        }
        var chatHistory = await chatHistoryProvider.GetAgentHistoryAsync(userMessage.UserId);
        var thread = new ChatHistoryAgentThread(chatHistory);
        try
        {
            var fullMessage = "";
            await foreach (ChatMessageContent response in agent.InvokeAsync(
            new ChatMessageContent(AuthorRole.User, userMessage.Message), thread))
            {
                fullMessage += response.Content;
            }
            return Results.Ok(new
            {
                message = fullMessage,
            });
        }
        catch (TaskCanceledException)
        {
            return Results.StatusCode(408);
        }
    }).AddEndpointFilter<ValidationFilter<UserChatMessage>>();
```

Code Examples

Semantic Kernel - Plugins

```
[KernelFunction("get_location")]
    [Description("Gets the current location and status of buses operating on the specified
line number.")]
    [return: Description(
            "Returns detailed information about buses currently operating on the specified
line, including position, speed, direction, and destination.")]
    private async Task<List<LinkkiLocationDetails>> GetLocationAsync(
        [Description("The bus line number or name to search for.")]
        string lineName)
    {
        var query = _locationContainer.GetItemLinqQueryable<LinkkiLocation>()
            .Where(l => l.Type == "bus"
            && l.Line.Name.ToLower() == lineName.ToLower().Trim());

        var details = new List<LinkkiLocationDetails>();
        using var iterator = query.ToFeedIterator();
        while (iterator.HasMoreResults)
        {
            details.AddRange((await iterator.ReadNextAsync())
            .Select(item => new LinkkiLocationDetails
            {
                Id = item.Id,
                Longitude = item.Location.Position.Longitude,
                Latitude = item.Location.Position.Latitude,
                Speed = item.Vehicle.Speed,
                Bearing = item.Vehicle.Bearing,
                Headsign = item.Vehicle.Headsign,
                TripId = item.Line.TripId,
                Direction = item.Line.Direction,
                LineName = item.Line.Name,
                LicensePlate = item.Vehicle.LicensePlate
            }));
        }
        return details;
    }
```

Code Examples

Semantic Kernel - Plugins

```
[KernelFunction("filter_bus_lines_on_map")]
[Description("Filters the bus lines shown on the map.")]
private async Task FilterBusLinesOnMapAsync(
    [Description("The bus line names to filter on the map")] List<string> lineNames)
{
    try
    {
        var userId = _httpContextAccessor.HttpContext?.Items["userId"] as string;
        await _webPubSubServiceClient.SendToUserAsync(userId,
            RequestContent.Create(new WebSocketEvent()
            {
                Type = "filter-bus-lines",
                Data = lineNames
            }), ContentType.ApplicationJson);
    }
    catch (Exception ex)
    {
        _logger.LogError(ex, "Failed to publish locations to WebPubSub Hub.");
    }
}
```

Code Examples

Semantic Kernel - Plugins

```
[KernelFunction("get_closest_bus_stops")]
[Description(
    "Gets the closest bus stops to the given location and within the given distance. The
default distance is 300 meters.")]
[return: Description("Returns the bus stops.")]
private async Task<List<BusStopLocationDetails>> GetClosestBusStopAsync(double longitude,
double latitude, double distance = 300)
{
    var query = _locationContainer.GetItemLinqQueryable<BusStopLocation>()
        .Where(s => s.Type == "stop")
        .Where(s => s.Location.Distance(new Point(longitude, latitude)) < distance)
        .Select(s => new BusStopLocationDetails
        {
            Name = s.Name,
            Location = s.Location,
            Distance = s.Location.Distance(new Point(longitude, latitude))
        });
    using var iterator = query.ToFeedIterator();
    var busStops = new List<BusStopLocationDetails>();
    while (iterator.HasMoreResults)
    {
        busStops.AddRange(await iterator.ReadNextAsync());
    }

    return busStops;
  }
```

Azure Web PubSub

```
builder.Services.AddWebPubSub(o =>
        o.ServiceEndpoint =
            new WebPubSubServiceEndpoint(new
Uri(builder.Configuration["WebPubSub:Endpoint"]!), azureCredential))
    .AddWebPubSubServiceClient<LinkkiHub>();
```

```
await _webPubSubServiceClient.SendToAllAsync(
RequestContent.Create(new WebSocketEvent()
{
    Type = "linkki-location",
    Data = locations.Select(location =>
        new
        {
            id = location.Id,
            line = location.Line.Name,
            coordinates = location.Location.Position.Coordinates,
            bearing = location.Vehicle.Bearing,
        })
}), ContentType.ApplicationJson);
```

```
var userId = _httpContextAccessor.HttpContext?.Items["userId"] as string;
await _webPubSubServiceClient.SendToUserAsync(userId,
    RequestContent.Create(new WebSocketEvent()
    {
        Type = "show-bus-stop",
        Data = new
        {
            name = busStopName,
            coordinates = new[] { longitude, latitude }
        }
    }), ContentType.ApplicationJson);
```

Lesson Learned

Insights

- Semantic Kernel is a powerful tool and is easy to use once you have set it up
- The plugins are amazing, and the way AI Model uses them feels like magic
- Azure Web PubSub can be a key component in building next-level AI applications by enabling real-time communication and interactivity

- You need to spend time designing prompts
  - Prompts (instructions) play a crucial role in AI. It is important to write clear and descriptive prompts. With good prompts, you can succeed and have AI that truly provides value to users. This applies to both system messages and function descriptions in plugins.
- Think about how many functions you have, their parameters, and their responses to help the AI Model pick the right one.
- Chat history is essential for maintaining context and enable better outputs for user
  - Semantic Kernel does not have native support for persisting chat history to a database. You need to write your own implementation.

- Access to high-quality data is essential for building AI Agents that provide real value for users and business
- Use the OpenAPI specification for building high-quality, AI- and developer-friendly API.
- Semantic Kernel OpenAPI plugin can use your OpenAPI specification without having any custom code

- AI usage is billed per token
  - Example: GPT-4o-mini cost 0.14€ per 1M tokens
- How many tokes does a chat session uses?
  - Semantic Kernel has native support for OpenTelemetry (metrics, logs, traces)
- Monitor usage& costs
  - Use Azure Application Insight for detailed app monitoring
  - Or use the Azure AI Foundry Metrics dashboard
- Set a budget to avoid surprise Azure costs!

Questions?

Thank you!