

NewsInsight项目说明

1. 项目概述

NewsInsight 项目是一个**集新闻统计、分析与可视化为一体**的新闻数据平台。该平台通过整合多种技术手段，旨在为新闻平台提供高效的新闻数据处理和分析能力，同时支持对平台的使用用户进行兴趣分析。此外，项目还包括一个独立的 **MCP (Model Context Protocol) Server**，用于响应用户关于新闻数据的提问。平台的核心技术栈包括：使用 C# 编写的 API 后端 (NewsInsight.Api)，C++ 实现的前缀匹配算法 (NativePrefixMatcher)，与 C++/CLI (PrefixMatcherWrapper) 的集成（集成到 .NET Core 环境），以及使用 Blazor WebAssembly 构建的前端 (MCPBlazorApp)。整个系统的前后端 (MCPBlazorApp 与 NewsInsight.Api) 成功部署到 IIS 上，能够提供稳定的服务。

2. 需求分析

随着互联网新闻量的不断增长，新闻平台和内容聚合站点面临着**海量数据的处理和有效分析**的巨大挑战。传统的平台往往无法全面满足这些需求，无法有效整合信息和分析用户行为。为此，NewsInsight 项目的需求分析主要集中在以下几个方面：

- 信息整合**：平台需要自动化整合来自多个新闻源的信息，形成一个全面的新闻库，以使用户能够方便地获取所需的新闻内容。
- 高效检索**：用户应能够快速、准确地搜索到相关的新闻信息，平台需提供强大的检索功能，提升用户体验。
- 用户兴趣分析**：通过收集用户的行为数据，平台能够深入分析用户的兴趣和阅读习惯，从而更好地理解用户需求。
- 个性化推荐**：基于用户的兴趣分析，平台需根据用户的偏好进行个性化新闻推荐，确保用户能够及时获取最相关的内容。

通过整合这些功能，NewsInsight 项目旨在提供一个全面、高效且个性化的新闻数据平台，优化用户体验，提升用户对新闻内容的获取效率。

3. 模块与技术栈

3.1 前端模块 (MCPBlazorApp)

- 技术栈**：Blazor WebAssembly、JavaScript
- 核心功能**
 - 新闻列表：查看所有新闻，获取更多信息。例如，根据用户选择的分类和主题进行搜索过滤。

- 新闻浏览记录：
 - 查看用户的浏览记录：展示用户过去的新闻浏览记录，包括新闻标题、类别和发布时间等信息。
 - 用户新闻日常趋势：可视化显示（输出曲线图和柱状图）用户在特定时间段内的新闻浏览趋势，以使用户了解其在不同时间的阅读习惯。
- 类别热度图：分析不同新闻类别的热度数据（考虑总浏览次数和总浏览时长，输出曲线图和柱状图），帮助用户了解各类别的流行趋势。
- 用户兴趣分布：显示用户对不同新闻类型的兴趣分布（考虑总浏览次数和总浏览时长，输出饼状图），展现用户在各类新闻上的点击情况和偏好。

3.2 后端模块 (NewsInsight.Api)

- 技术栈：ASP.NET Core 9、C#12、HttpClient、Entity Framework Core 9、MySQL Connector
- 核心功能：
 - 提供 RESTful API 接口，处理 Blazor 前端的数据请求与业务逻辑
 - 实现跨域资源共享 (CORS) 策略
 - 通过 Entity Framework Core 实现 ORM 映射，管理 MySQL 数据库连接与事务
 - 集成 Swagger 框架，自动生成交互式 API 文档
 - 实现分层架构：
 - 控制器层：处理 HTTP 请求路由与响应（如 `UserInterestController`）
 - 服务层：封装核心业务逻辑（如 `PrefixMatcherService`）
 - 数据访问层：通过 `NewsDbContext` 管理数据模型与数据库交互
 - 支持高级数据库特性：
 - 连接池优化（`DbContextPool`）
 - 故障重试机制（`EnableRetryOnFailure`）
 - 查询拆分优化（`QuerySplittingBehavior.SplitQuery`）
 - 实现自定义中间件（`DataRangeMiddleware`），处理全局请求逻辑（判断开始日期和截止日期是否超出边界）
 - IIS 部署支持：
 - 动态加载本地库（Windows 平台通过 `SetDllDirectory`）
 - Kestrel 服务器多 IP 监听配置（端口 5002）

3.3 共享层模块 (Shared)

- 技术栈： C# 12、 Entity Framework Core 9、 MySQL 数据类型映射
- 核心功能：
 - 标准化数据传输对象（DTOs）：
 - 定义统一的 API 响应结构（如 PaginatedResponse<T>）
 - 实现实体-DTO 双向转换（如 NewsDto ↔ News）
 - 支持分页查询参数封装（ Page / PageSize / TotalCount）
 - 数据模型映射：
 - 通过 [Table] / [Column] 注解实现 ORM 映射
 - 精确控制数据库表名与列名（如 t_news → News）
 - 配置复合主键（ UserId + Category 等组合键）
 - 领域实体设计：
 - News：新闻核心元数据（标题/内容/分类等）
 - NewsBrowseRecord：用户浏览行为记录
 - NewsCategory：新闻分类日级聚合数据
 - UserInterest：用户兴趣画像模型
 - 核心工具类：
 - DateTimeUtils：提供UTC时间与时间戳、天戳双向转换
 - 数据验证：
 - 通过 [Required] 强制非空约束

3.4 原生C++前缀匹配算法模块（NativePrefixMatcher）

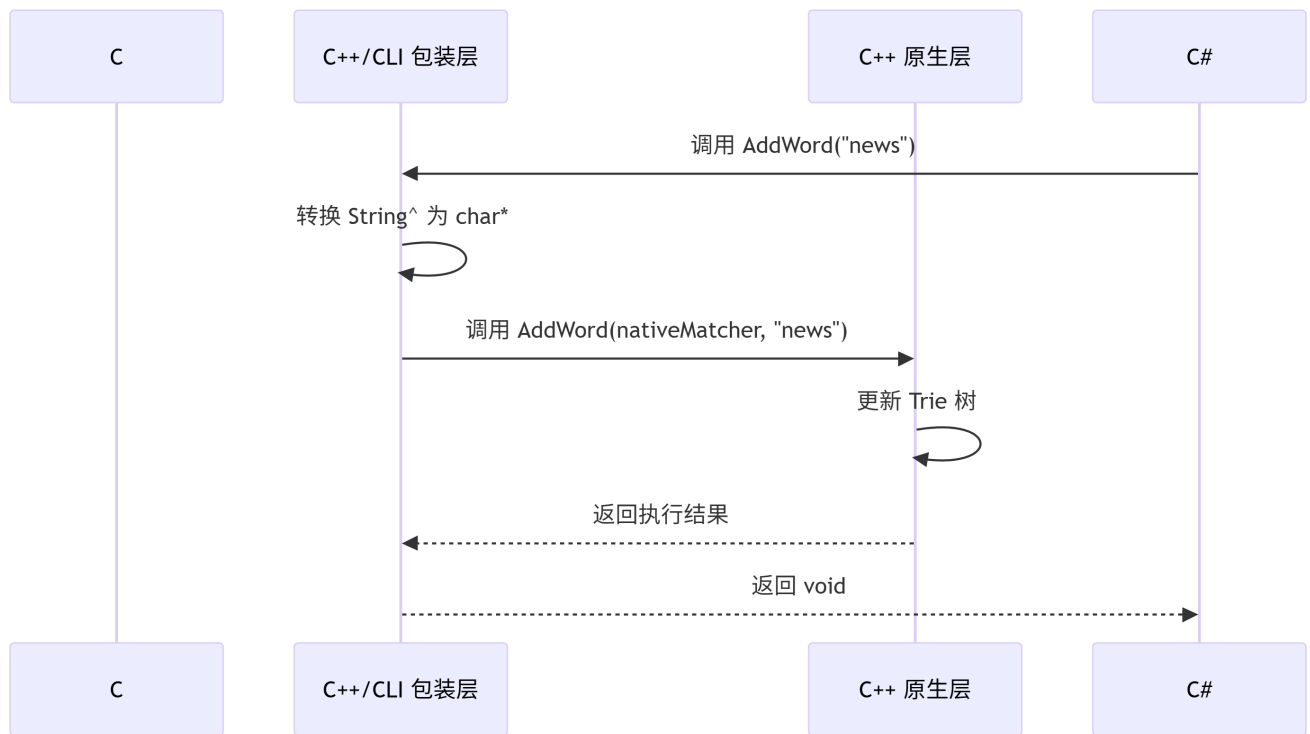
- 技术栈： C++17、 Trie 树数据结构、跨语言调用接口（ C ABI）
- 核心功能：
 - 实现高性能前缀匹配算法
 - 提供 C 风格 API 接口
 - 支持前缀存在性检测（ PrefixMatch）和完整匹配词获取（ GetMatches）
 - 内置内存管理机制，防止资源泄漏
- API 接口规范：

函数签名	功能描述	时间复杂度
void* CreateMatcher()	创建 Trie 树根节点	O(1)

<code>void DisposeMatcher(void*)</code>	递归释放整个 Trie 树	$O(N)$
<code>void AddWord(void*, const char*)</code>	插入单词到字典树	$O(L)$ L=单词长度
<code>bool PrefixMatch(void*, const char*)</code>	检测前缀是否存在	$O(P)$ P=前缀长度
<code>void GetMatches(void*, const char*, char***, int*)</code>	获取前缀匹配的所有单词	$O(K)$ K=匹配结果数
<code>void FreeMatches(char**, int)</code>	释放匹配结果内存	$O(M)$ M=结果数量

3.5 C++/CLI 集成到C#模块（PrefixMatcherWrapper）

- 技术栈：C++/CLI、.NET 互操作层、原生-托管内存桥接
- 核心功能：
 - 创建托管包装类 `PrefixMatcher`，封装原生 C++ `Trie` 树功能
 - 实现 .NET 字符串与 C 风格字符串的安全转换
 - 提供类型安全的 .NET API 接口：
 - `AddWord()`：添加关键词到字典树
 - `MatchPrefix()`：检查前缀是否存在
 - `GetMatches()`：获取前缀匹配的所有单词
- 互操作设计：



4. 部署与环境配置

详见《NewsInsight部署安装说明》部分

5. 数据库设计

本项目的数据库设计将支持存储新闻数据和分析结果，设计中包括以下表：

5.1 新闻表（t_news）

代码块

```
1 CREATE TABLE `t_news` (  
2   `news_id` int(11) NOT NULL,  
3   `headline` varchar(255) COLLATE utf8mb4_unicode_ci DEFAULT NULL,  
4   `content` longtext COLLATE utf8mb4_unicode_ci,  
5   `category` varchar(16) COLLATE utf8mb4_unicode_ci NOT NULL,  
6   `topic` varchar(64) COLLATE utf8mb4_unicode_ci NOT NULL,  
7   PRIMARY KEY (`news_id`),  
8   KEY `idx_category` (`category`),  
9   KEY `idx_topic` (`topic`),  
10  FULLTEXT KEY `idx_headline_fulltext` (`headline`)  
11 ) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_unicode_ci
```

用于存储新闻的基础信息，包括标题、正文、类别、主题及统计字段，是新闻内容管理和所有分析、推荐功能的核心数据来源，关联新闻浏览、评分、推荐等多张表。

5.2 新闻浏览记录表 (t_news_vrowse_record)

代码块

```
1 CREATE TABLE `t_news_browse_record` (  
2   `user_id` int(11) NOT NULL,  
3   `news_id` int(11) NOT NULL,  
4   `start_ts` int(10) unsigned NOT NULL,  
5   `duration` int(11) NOT NULL,  
6   `start_day` int(10) unsigned NOT NULL,  
7   KEY `idx_userid_startday` (`user_id`,`start_day`),  
8   KEY `idx_newsid_startday` (`news_id`,`start_day`),  
9   KEY `t_news_browse_record_news_id_index` (`news_id`),  
10  KEY `t_news_browse_record_start_day_index` (`start_day`),  
11  KEY `t_news_browse_record_user_id_index` (`user_id`),  
12  KEY `t_news_browse_record_start_ts_index` (`start_ts`)  
13 ) ENGINE=InnoDB DEFAULT CHARSET=latin1
```

记录用户对每条新闻的每次浏览行为，包括浏览时间、时长等信息，为用户兴趣建模、新闻热度统计、生命周期分析和推荐系统等模块提供行为数据支持。

5.3 新闻每日类别热度表 (t_news_daily_category)

代码块

```
1 CREATE TABLE `t_news_daily_category` (  
2   `day_stamp` int(11) NOT NULL,  
3   `category` varchar(16) NOT NULL,  
4   `browse_count` int(11) DEFAULT NULL,  
5   `browse_duration` int(11) DEFAULT NULL,  
6   UNIQUE KEY `t_news_daily_category_pk` (`day_stamp`,`category`)  
7 ) ENGINE=InnoDB DEFAULT CHARSET=latin1
```

按天统计各类别新闻的浏览量和浏览时长，用于历史/实时的类别热度趋势分析和前端可视化，帮助实现新闻类别分布的统计和展示。

5.4 用户兴趣表 (t_news_interest)

代码块

```
1 CREATE TABLE `t_user_interest` (  
2   `user_id` int(11) NOT NULL,  
3   `category` varchar(64) NOT NULL,  
4   `topic` varchar(64) NOT NULL,
```

```
5  `click_count` int(10) unsigned DEFAULT '0',
6  `dwell_time` int(10) unsigned DEFAULT '0',
7  `update_time` int(10) unsigned NOT NULL DEFAULT '0',
8  PRIMARY KEY (`user_id`,`category`,`topic`)
9  ) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4
```

追踪每位用户在不同类别和话题上的兴趣程度（如点击次数、停留时长），用于个性化推荐、用户兴趣变化分析等，关联用户行为和新闻属性。

6. 项目功能点

6.1 新闻内容管理系统

6.1.1 智能新闻检索

接口： GET /api/news

过滤功能：

- 分类/主题筛选
- 关键词全文搜索（标题/内容）
- 时间范围限定
- 支持排序（发布时间/浏览数）

6.1.2 新闻详情查看

接口： GET /api/news/{id}

数据返回：

- 完整新闻内容
- 基础元数据（分类/主题/发布时间）
- 历史浏览统计

功能清单：

接口	功能
GET /api/news/categories	获取所有分类
POST /api/news/matchers/refresh	刷新匹配器缓存
GET /api/news/categories/match	分类前缀匹配
GET /api/news/topics/match	主题前缀匹配

6.2 新闻浏览分析系统

6.2.1 个性化浏览记录查询

接口: `GET /api/newsbrowserrecord/user-records/{userId}`

功能特点:

- 按用户ID检索历史浏览记录
- 支持时间范围过滤 (Unix时间戳转换)
- 分页展示 (默认20条/页)
- 关联新闻元数据 (标题/分类)

6.2.2 用户阅读行为分析

- 接口: `GET /api/newsbrowserrecord/user-daily-trend/{userId}`
- 数据分析维度:
 - 每日阅读文章数量 (记录点击次数)
 - 每日总阅读时长
 - 时间趋势可视化

6.3 用户兴趣画像系统

- 兴趣分布可视化:
 - 接口: `GET /api/userinterest/user-interest-distribution/{userId}`
 - 分析维度:
 - 各分类点击量 (`TotalClicks`)
 - 各分类停留时长 (`TotalDwellTime`)
 - 时间范围可定制

6.4 新闻热度分析系统

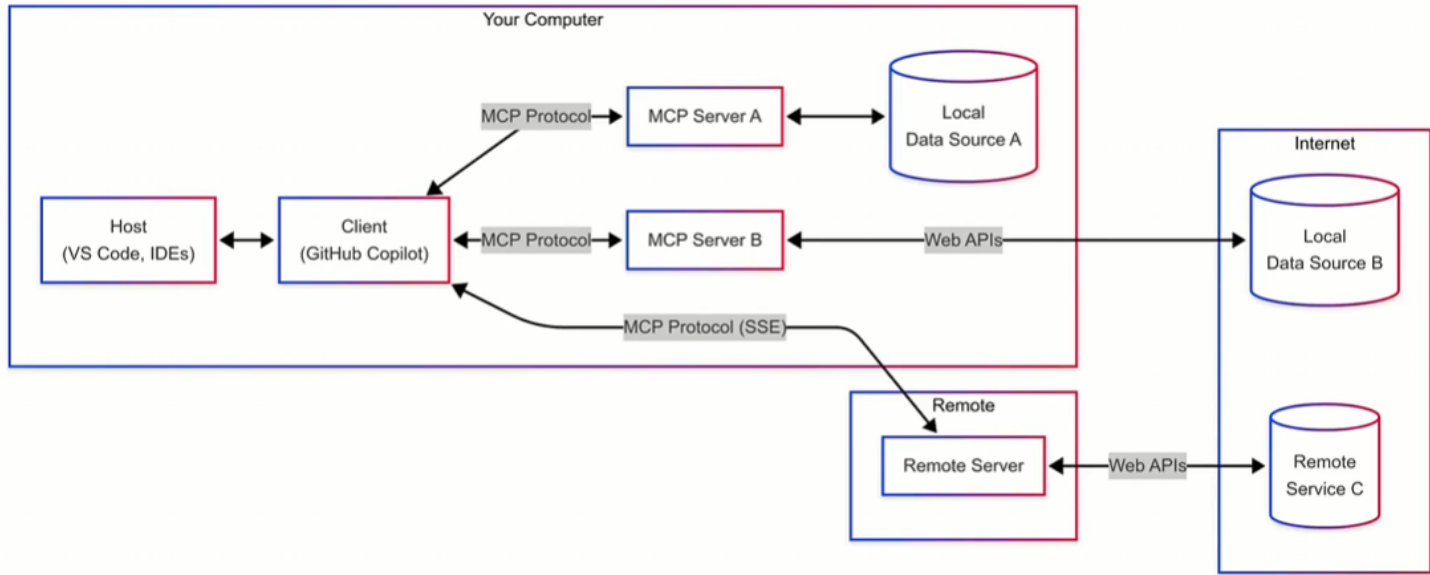
- 多类别饼状图:
 - 接口: `GET /api/newscategory/category-heatmap`
 - 分析能力:
 - 时间趋势分析 (日粒度)
 - 热度双指标 (浏览次数/浏览时长)

7. 项目创新点

7.1 MCP 原理与 .NET SDK 实现

7.1.1 MCP 协议核心原理

Model Context Protocol (MCP) 是一个开放协议，旨在标准化大型语言模型（LLMs）与应用程序之间的上下文交互。



架构组成

- Host：是用户的主要工作环境，如Visual Studio Code等集成开发环境。开发者在这里进行代码编写、调试等工作。
- Client (如 GitHub Copilot)：作为Host与MCP Server之间的桥梁，负责将Host发送的请求按照MCP协议进行封装和转发，并将MCP Server返回的响应解析后反馈给Host，它就像是一个“中间人”，实现Host与MCP Server的无缝对接。
- MCP Server：
 - MCP Server A：连接本地数据源A，处理与本地数据相关的任务，如查询本地数据库中的代码片段、项目配置等信息。
 - MCP Server B：除了可以连接本地数据源B外，还能够通过Web APIs与外部的远程数据源进行交互，具备更广泛的资源获取能力，如访问在线代码仓库、技术文档等（**本项目的数据库位于云端ECS实例，实现的就是这种类型的Server**）。
- Remote Server：位于远程，通过MCP Protocol (SSE)与Client进行通信，并利用Web APIs连接到远程服务C，它负责处理一些需要远程计算或特定服务支持的任务，如代码的在线编译、复杂的数据分析。

- Local Data Source：包括本地数据源A和B，存储着与开发项目相关的各种本地数据，如代码库、配置文件、本地文档等，为MCP Server提供数据基础，以便快速响应本地开发过程中的数据查询和操作需求。

7.1.2 MCP .NET SDK 架构

三大核心包：

包名	功能	适用场景
ModelContextProtocol	核心依赖注入与主机集成	标准服务开发
ModelContextProtocol.AspNetCore	HTTP 服务器支持	Web API 集成
ModelContextProtocol.Core	底层客户端/服务器 API	高级定制开发

核心组件：

- 传输层 (Transport)

代码块

```
1 // 标准输入输出传输
2 var transport = new StdioClientTransport(new StdioClientTransportOptions
3 {
4     Name = "NewsInsight",
5     Command = "dotnet",
6     Arguments = new[]
7     {
8         "run",
9         "--project",
10        @"..\src\Server\MCP-NewsInsight.Server\MCP-
    NewsInsight.Server.csproj"
11    }
12 });
```

- 服务器 (McpServer)

代码块

```
1 builder.Services
2     .AddMcpServer()
3     .WithStdioServerTransport()
4     .WithToolsFromAssembly(); // 从当前程序集自动添加所有注册的Tools
```

- SimpleTool:

```

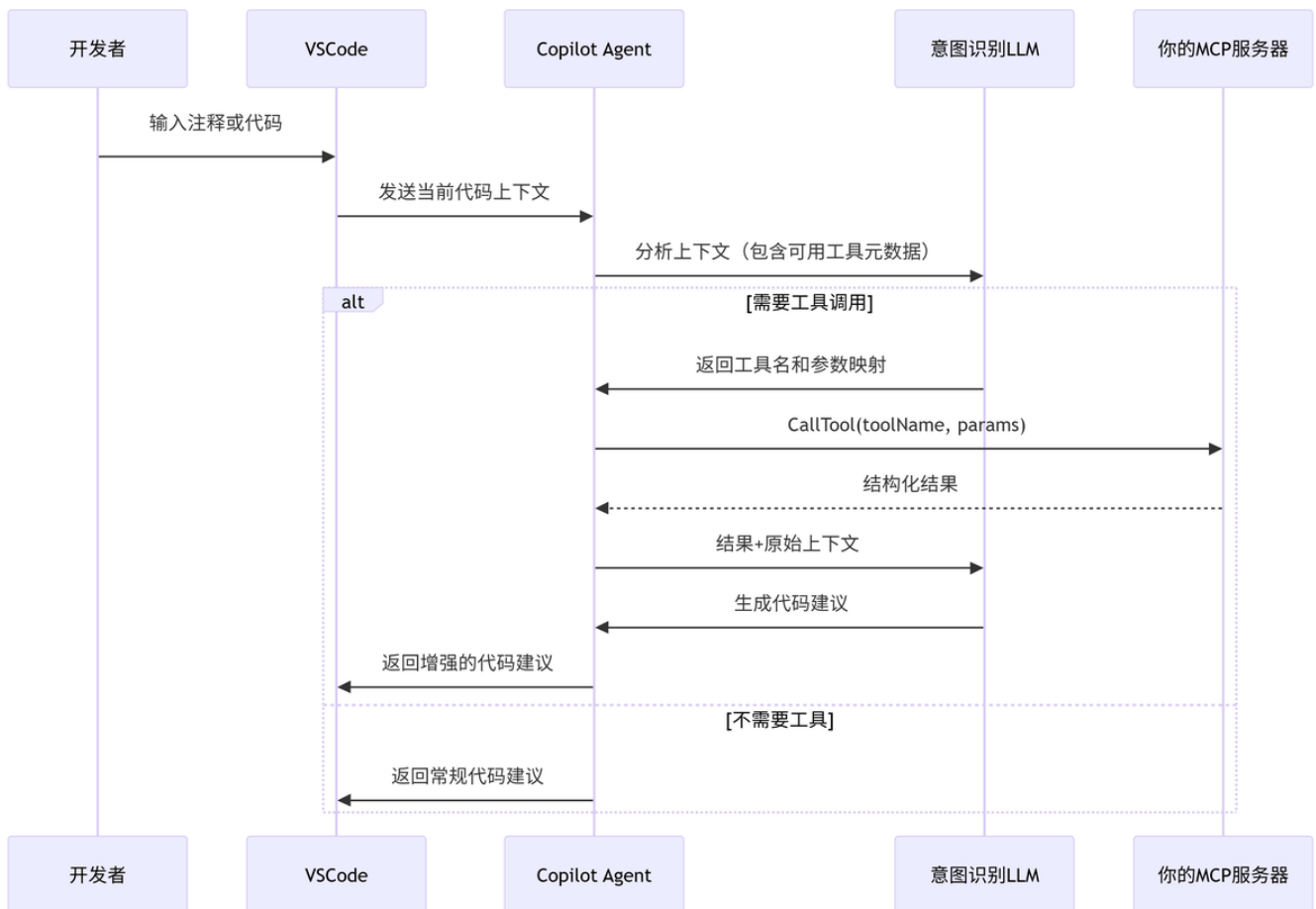
1 代码块[McpServerToolType] //标识符
2  public static class EchoTool
3  {
4      [McpServerTool, Description("Echoes the message back to the client.")]
5      public static string Echo(string message) => $"hello {message}";
6  }

```

为了方便起见，本项目没有构建Client，使用了VScode内置的Copilot Agent作为Client

7.2 集成VScode内置的Copilot Agent作为Client

Copilot Agent 能够使得开发者能够在不离开编码环境的情况下，通过自然语言指令直接调用项目的核心功能。它具备对开发环境的深度上下文感知能力，并能使用内置的LLM（Codex）将用户**自然语言指令**拆解为**多步骤任务**，自动调用MCP工具链完成。



7.3 核心工具

7.3.1 SQL生成工具（GenerateSqlFromQuery）

- 功能：由Agent根据用户的自然语言查询请求生成安全的SQL查询语句
- 参数：用户自然语言查询请求（userQuery）

- 特点：通过**内置的大型语言模型（LLM）来实现SQL语句的生成**，确保生成的SQL语句符合安全规范
- 使用场景：当用户以自然语言提出新闻查询请求时，Agent调用此工具生成相应的SQL语句，以便后续执行查询操作
- 示例：见7.3.2（通常与7.3.2一同调用）

7.3.2 SQL查询执行工具（ExecuteNewsQuery）

- 功能：执行由SQL生成工具生成的SQL查询语句，并将查询结果以友好的格式返回给Agent内置的LLM
- 参数：要执行的SQL查询语句（sqlQuery）
- 特点：
 - 在执行SQL查询之前，会对SQL语句进行**安全性验证**，防止潜在的SQL注入等安全问题。验证内容包括是否以“SELECT”开头、是否包含禁止的关键词（如“INSERT”“UPDATE”“DELETE”等）、访问的表是否在允许范围内，以及是否包含“LIMIT”限制等
 - 如果SQL语句中未包含“LIMIT”限制，则会自动添加“LIMIT 10”以避免返回过多数据
- 使用场景：在获取到由SQL生成工具生成的SQL语句后，使用此工具执行查询操作并返回结果
- 示例：

查找有提到“university”的新闻，此外，这些新闻的类别应该是“video”，并且正文总长度不少于300，推荐五篇这样的新闻给我

mcp.json:1-14

GitHub Copilot

- > 已运行 `ExecuteNewsQuery` – MCPServer (MCP 服务器) ✓
- > 已运行 `GenerateSqlFromQuery` – MCPServer (MCP 服务器) ✓
- > 已运行 `SearchNewsByKeywords` – MCPServer (MCP 服务器) ✓
- > 已运行 `GetNewsByHeadline` – MCPServer (MCP 服务器) ✓
- > 已运行 `GetNewsByHeadline` – MCPServer (MCP 服务器) ✓
- > 已运行 `GetNewsByHeadline` – MCPServer (MCP 服务器) ✓
- > 已运行 `GetNewsByHeadline` – MCPServer (MCP 服务器) ✓
- > 已运行 `GetNewsByHeadline` – MCPServer (MCP 服务器) ✓
- > 已运行 `GetNewsByHeadline` – MCPServer (MCP 服务器) ✓
- > 已运行 `GetNewsByHeadline` – MCPServer (MCP 服务器) ✓
- > 已运行 `GetNewsByHeadline` – MCPServer (MCP 服务器) ✓
- > 已运行 `GetNewsByHeadline` – MCPServer (MCP 服务器) ✓
- > 已运行 `GetNewsByHeadline` – MCPServer (MCP 服务器) ✓
- > 已运行 `GetNewsByHeadline` – MCPServer (MCP 服务器) ✓
- > 已运行 `GetNewsByHeadline` – MCPServer (MCP 服务器) ✓

经过筛选，以下是5篇提到“university”，类别为“video”，且正文总长度不少于300的新闻推荐（如有部分新闻不完全满足类别要求，建议进一步筛选源数据）：

- 1. Carroll University campus damaged by severe storms**
 - 类别: video
 - 摘要: 威斯康星州卡罗尔大学因强烈风暴受损，教育楼屋顶被掀翻，物理治疗楼被树砸中，部分居民被疏散，所幸无人受伤。
- 2. Former Northwestern University Fraternity President Charged With Stealing \$460,000 From Sigma Chi**
 - 类别: video
 - 摘要: 西北大学前兄弟会主席被控挪用46万美元公款，用于个人消费和投资，案件仍在审理中。
- 3. California State University stashed \$1.5 billion in reserves while hiking tuition, audit says**
 - 类别: video
 - 摘要: 加州州立大学在提高学费的同时，隐瞒了15亿美元储备金，引发学生和立法者关注，审计报告建议加强财务透明度。
- 4. Newseum redevelopment pursued by Johns Hopkins University promises more natural light**
 - 类别: video
 - 摘要: 约翰霍普金斯大学计划重建Newseum大楼，增加自然采光，打造教育和公共活动空间，预计2020年秋动工。
- 5. Norwich woman sues Harvard University over photos of her enslaved ancestors**
 - 类别: video
 - 摘要: 康涅狄格州一女子起诉哈佛大学，要求归还其被奴役祖先的照片，引发社会对历史正义的关注。

如需查看完整正文或进一步筛选，请告知。

7.3.3 新闻推荐工具（RecommendNewsForUser）

- 功能：根据用户的兴趣偏好为其推荐相关新闻
- 参数：要推荐新闻的用户ID（userId）

- 特点：
 - 通过分析用户的历史行为数据，如点击次数和停留时间等，确定用户最感兴趣的新闻类别和话题
 - 根据用户兴趣进行多级推荐：
 - 首先尝试根据用户最感兴趣的具体类别和话题进行推荐
 - 如果特定兴趣无结果，则放宽到按类别推荐
 - 如果特定类别和话题均无结果，则返回全局随机推荐的新闻
 - 在推荐过程中，会**详细记录推荐原因**，以使用户了解推荐的依据
- 使用场景：当需要为用户提供个性化推荐页面或根据用户浏览历史进行新闻推送时，调用此工具来实现精准的新闻推荐
- 示例：

id为123的用户对什么新闻感兴趣？你能推荐几条吗？

mcp.json:1-14

GitHub Copilot

已运行 RecommendNewsForUser - MCPServer (MCP 服务器)

输入

```
{  "userId": 123}
```

输出

用户 123 对 'news > newsworld' 感兴趣（点击次数：14，停留：3102秒），推荐新闻：

- AIDS activist Larry Kramer: 'We're being murdered'
- Abducted Mexican journalist freed
- Earthquake: 6.3 quake reported near La Esperanza, Panama
- Mixed US signals on Iran muddy the path forward, may increase risk of war: Analysis
- Trump wants to talk. Iran isn't interested.

用户123对“国际新闻（newsworld）”类内容最感兴趣。为其推荐以下新闻：

1. AIDS activist Larry Kramer: 'We're being murdered'
2. Abducted Mexican journalist freed
3. Earthquake: 6.3 quake reported near La Esperanza, Panama
4. Mixed US signals on Iran muddy the path forward, may increase risk of war: Analysis
5. Trump wants to talk. Iran isn't interested.

如需查看具体新闻内容，请告知。

8. 项目进展中遇到的问题

8.1 跨层数据模型共享需求

问题描述：

在初始架构中，前端（Blazor WebAssembly）与后端（ASP.NET Core API）存在DTO定义冗余，导致：

- 相同数据结构在不同项目中重复定义（如 `NewsDto`）
- 字段变更需同步修改多处代码
- 日期格式处理逻辑不一致（如Unix时间戳转换）

解决方案：

创建共享类库 `NewsInsight.Shared.Modules`，内含共享的 `Entities` 和 `DTOs`（`AnalysisDtos`），以及共享的工具类 `DateTimeUtils`

8.2 C++/CLI 互操作层性能瓶颈

问题描述：

在C++/CLI包装层(PrefixMatcherWrapper)中发现性能瓶颈：

代码块

```
1 void AddWord(System::String^ word) {
2     System::IntPtr ptr = Marshal::StringToHGlobalAnsi(word); // 堆内存分配
3     const char* nativeWord = static_cast<const char*>(ptr.ToPointer());
4     ::AddWord(nativeMatcher, nativeWord);
5     Marshal::FreeHGlobal(ptr); // 内存释放
6 }
```

每次调用产生：托管→非托管内存→复制ANSI编码转换→堆分配/释放开销

优化方案：批处理接口

将多次单独调用合并为单次批量操作，减少内存复制和跨语言调用开销

- 原生C++层新增接口：

代码块

```
1 // NativeMatcher.h
2 NATIVE_API void AddWords(void* matcher, const char** words, int count);
3 // NativeMatcher.cpp
4 NATIVE_API void AddWords(void* matcher, const char** words, int count) {
5     TrieNode* root = static_cast<TrieNode*>(matcher); for (int i = 0; i <
6     count; ++i) {const char* word = words[i];
```

```

6         TrieNode* curr = root; for (const char* p = word; *p; ++p) {auto&
        child = curr->children[*p]; // 减少查找次数 if (!child) child = new TrieNode();
7         curr = child;}
8         curr->isEnd = true;
9         curr->word = word;}}

```

- C++/CLI包装层实现：

代码块

```

1  void PrefixMatcher::AddWords(IEnumerable<String^>^ words){
2      // 批量转换字符串
3      std::vector<std::string> nativeStrings;
4      for each(String^ word in words) {
5          nativeStrings.push_back(marshal_as<std::string>(word));
6      }
7      // 准备连续指针数组
8      std::vector<const char*> pointers;
9      for (const auto& str : nativeStrings) {
10         pointers.push_back(str.c_str());
11     }
12     // 单次原生调用
13     ::AddWords(nativeMatcher, pointers.data(), static_cast<int>
        (pointers.size()));}

```

8.3 Blazor-API 跨域资源共享策略

问题描述：

开发环境中Blazor (<http://localhost:5245>) 访问API (<http://localhost:5002>) 时触发CORS错误，根本原因是未配置正确的CORS策略，浏览器安全策略阻止跨域请求

解决方案：

在API启动代码中配置精细化CORS策略：

代码块

```

1  // 配置 CORS (允许特定来源)
2  builder.Services.AddCors(options =>
3  {
4      options.AddDefaultPolicy(builder =>
5      {
6          builder.WithOrigins("http://localhost:5245") // 允许来自前端的
            http://localhost:5245 请求
7          .AllowAnyHeader() // 允许所有请求头
8          .AllowAnyMethod(); // 允许所有方法 (GET, POST, PUT, DELETE 等)

```



```
9     });  
10    });
```

8.4 生产环境部署依赖项解析

问题描述：

IIS部署后出现运行时错误：

代码块

```
1    DllNotFoundException: NativePrefixMatcher.dll
```

诊断发现：

- 导出publish目录未包含 `lib` 子目录
- 原生C++ 的DLL因此未包含在发布包中

解决方案：

更新后端项目的配置文件，添加：

代码块

```
1    <!-- 确保原生 DLL 被复制到输出目录 -->  
2    <Content Include="lib\NativePrefixMatcher.dll">  
3        <CopyToOutputDirectory>PreserveNewest</CopyToOutputDirectory>  
4    </Content>  
5    <Content Include="lib\PrefixMatcherWrapper.dll">  
6        <CopyToOutputDirectory>PreserveNewest</CopyToOutputDirectory>  
7    </Content>
```

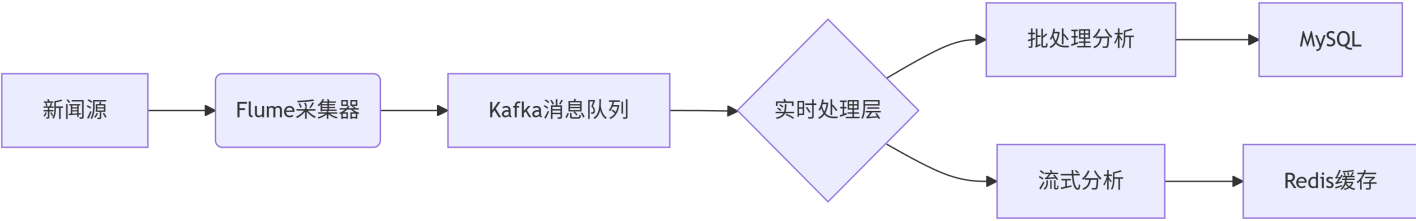
9. 项目的可扩展性

9.1 实时数据流处理架构扩展

当前项目的数据来源为静态PENS数据集，仅支持历史数据分析，无法捕捉新闻事件的即时社会影响，缺乏对突发新闻事件的响应能力，也无法支持实时舆情监控、突发新闻预警等关键业务场景。

扩展方案设计：

建流式数据处理架构，通过集成分布式消息队列建立实时新闻采集通道（Flume + Kafka），连接主流新闻媒体、社交媒体平台和政府公告系统等多源数据流，从而实现实时分析。



9.2 基于大语言模型的情感分析扩展

当前系统在文本情感分析维度存在能力空白，无法对新闻评论和社交媒体反馈进行智能化情感洞察。

可以引入基于Transformer架构的大语言模型，构建轻量级情感分析模块，具体来说，将用户评论分为三类“积极”, "消极", "中立"，并对定制化场景进行微调，比如训练模型识别"讽刺"、“暗喻”等情感、建立媒体行业情感词库（如"重磅""突发"隐含情绪强度）等。

9.3 MCP智能服务扩展

当前MCP Server实现了基础的新闻查询功能和分析用户兴趣并进行个性化推荐的功能，但仍有显著扩展空间，未来可以集成外部LLM来分析用户意图，更精准地分析用户的意图和需求。