Умные ссылки

Архитектура и шаблоны проектирования

Меня хорошо видно & слышно?

Защита проекта
Тема: Умные ссылки

Некрасов Станислав

Разработчик .NET, C# и немного JS

План защиты

- Цель и задачи проекта
- Какие технологии использовались
- Что получилось
- Выводы
- Вопросы и рекомендации

Цель и задачи проекта

Цель проекта: Пользователь переходит по ссылке. В зависимости от набора правил, которые могут включать интервалы времени, местоположение пользователя, устпройство, браузер и др. система выполняет редирект на урл, соответствующий сработавшему правилу.

1. Управление правилами перенаправления.
2. Обогащение запроса: тип устройства, браузер и т.д.
3. Парсинг и обработка правил.
4. Аутентификация посредствам JWT, на основе public key, private key
5. Накопление статистики редиректов.

Какие технологии использовались

1. Проект написан на C# .NET 8
2. В качестве бэка ASP.NET Minimal API
3. Админка для управления пользователями Blazor (Web Assembly)
4. Хранилище данных MongoDB
5. Брокер RabbitMQ обертка MassTransit
6. Aspire .NET запуска зависимостей

Что получилось

https://github.com/NekrasovSt/otus-main-patterns-project

1. Из запроса извлекаются данные: IP, предпочитаемый язык, query-параметры, браузер, ос, тип устройства. Каждый элемент данных заполняется в отдельном классе *Filler имплементация IFiller, динамически загружается через DI

```
/// <summary>
/// Заполнитель запроса
/// </summary>
public interface IFiller
{
    /// <summary>
    /// Сформировать словарь дополнительных данных
    /// </summary>
    IReadOnlyDictionary<string, object> Fill();
}
```

2. Правила извлекаются из базы и компилируются в Expression

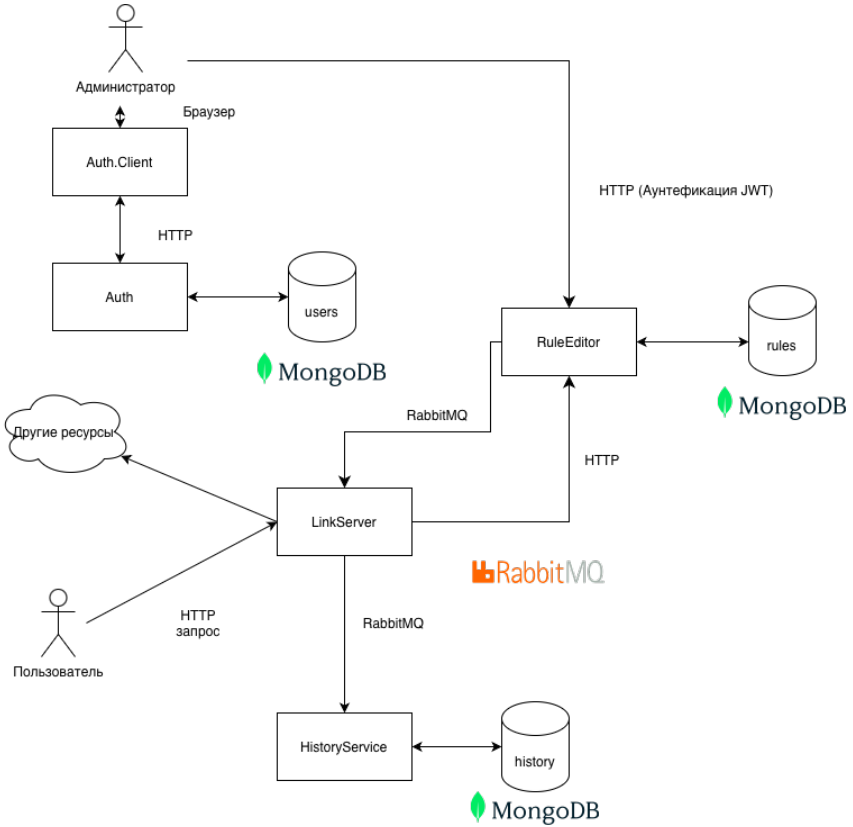
```
{
    "id": "69146d13ee48377e92f0704d",
    "name": "Демо правило 3",
    "order": 2,
    "link": "http://www.google.ru",
    "filterCondition": {
      "conditions": [],
      "field": "os",
      "operator": "=",
      "value": "Win 64"
    }
  }
```

```
public class DictionaryExpressionBuilder
{
    public Expression<Func<Dictionary<string, object>, bool>> BuildExpression(FilterCondition filter)
    {
        var parameter = Expression.Parameter(typeof(Dictionary<string, object>), "dict");
        var expression = BuildCondition(parameter, filter);
        return Expression.Lambda<Func<Dictionary<string, object>, bool>>(expression, parameter);
    }
  //…
}
```

3. Проверка правил.

```
foreach (var rule in compiledRules)
{
    if (rule.Predicate.Invoke(dictionary))
    {
        await _sendEndpointProvider.Publish(new RuleExecutedDto()
        {
            Date = DateTime.UtcNow,
            RuleId = rule.Id,
            RuleName = rule.Name,
            Url = rule.Link
        }, token);
        return rule.Link;
    }
}
```

Диаграмма проекта



Возможные сложности

1. Высокая нагрузка на LinkServer.
    - Решение: создание нескольких экземпляров LinkServer

2. Не равномерная нагрузка на разные экземпляры LinkServer
    - Решение: использование балансировщиков нагрузки.

3. Слишком частые запросы LinkServer на RuleEditor
    - Решение: использование стороннего кэша.

Выводы

1. Удалось использовать микросервистный подход и запуск в docker.
2. Удалось поработать с нереляционной базой данных.
3. Удалось применить принципы SOLID
4. Примеры такие подходы как middleware, DI

Вопросы и рекомендации

\+ если есть вопросы

— если вопросов нет

Спасибо за внимание!