LinksPlatform's Platform.Bot Class Library

1.1 ./csharp/Platform.Bot/Program.cs

```csharp
using csharp;
using Interfaces;
using Octokit;
using Platform.Exceptions;
using Platform.IO;
using Storage.Local;
using Storage.Remote.GitHub;
using System;
using System.Collections.Generic;

namespace Platform.Bot
{
    /// <summary>
    /// <para>
    /// Represents the program.
    /// </para>
    /// <para></para>
    /// </summary>
    internal class Program
    {
        /// <summary>
        /// <para>
        /// Main the args.
        /// </para>
        /// <para></para>
        /// </summary>
        /// <param name="args">
        /// <para>The args.</para>
        /// <para></para>
        /// </param>
        private static void Main(string[] args)
        {
            using var cancellation = new ConsoleCancellation();
            var argumentIndex = 0;
            var username = ConsoleHelpers.GetOrReadArgument(argumentIndex++, "Username", args);
            var token = ConsoleHelpers.GetOrReadArgument(argumentIndex++, "Token", args);
            var appName = ConsoleHelpers.GetOrReadArgument(argumentIndex++, "App Name", args);
            var databaseFileName = ConsoleHelpers.GetOrReadArgument(argumentIndex++, "Database
            ↪  file name", args);
            var fileSetName = ConsoleHelpers.GetOrReadArgument(argumentIndex++, "File set name
            ↪  ", args);
            var dbContext = new FileStorage(databaseFileName);
            Console.WriteLine($"Bot has been started. {Environment.NewLine}Press CTRL+C to
            ↪  close");
            try
            {
                var api = new GitHubStorage(username, token, appName);
                new ProgrammerRole(
                    new List<ITrigger<Issue>> {
                        new HelloWorldTrigger(api, dbContext, fileSetName),
                        new OrganizationLastMonthActivityTrigger(api),
                        new LastCommitActivityTrigger(api)
                    },
                    api
                ).Start(cancellation.Token);
            }
            catch (Exception ex)
            {
                Console.WriteLine(ex.ToStringWithAllInnerExceptions());
            }
        }
    }
}
```

1.2 ./csharp/Platform.Bot/ProgrammerRole.cs

```csharp
using Interfaces;
using Octokit;
using Storage.Remote.GitHub;
using System;
using System.Collections.Generic;
using System.Threading;

namespace Platform.Bot
{
    /// <summary>
    /// <para>
    /// Represents the programmer role.
    /// </para>
    /// <para></para>
    /// </summary>
    public class ProgrammerRole
    {
        /// <summary>
        /// <para>
        /// The git hub api.
        /// </para>
        /// <para></para>
        /// </summary>
        public readonly GitHubStorage GitHubAPI;

        /// <summary>
        /// <para>
        /// The minimum interaction interval.
        /// </para>
        /// <para></para>
        /// </summary>
        public readonly TimeSpan MinimumInteractionInterval;

        /// <summary>
        /// <para>
        /// The triggers.
        /// </para>
        /// <para></para>
        /// </summary>
        public readonly List<ITrigger<Issue>> Triggers;

        /// <summary>
        /// <para>
        /// Initializes a new <see cref="ProgrammerRole"/> instance.
        /// </para>
        /// <para></para>
        /// </summary>
        /// <param name="triggers">
        /// <para>A triggers.</para>
        /// <para></para>
        /// </param>
        /// <param name="gitHubAPI">
        /// <para>A git hub api.</para>
        /// <para></para>
        /// </param>
        public ProgrammerRole(List<ITrigger<Issue>> triggers, GitHubStorage gitHubAPI)
        {
            GitHubAPI = gitHubAPI;
            Triggers = triggers;
            MinimumInteractionInterval = gitHubAPI.MinimumInteractionInterval;
        }

        /// <summary>
        /// <para>
        /// Processes the issues using the specified token.
        /// </para>
        /// <para></para>
        /// </summary>
        /// <param name="token">
        /// <para>The token.</para>
        /// <para></para>
        /// </param>
        private void ProcessIssues(CancellationToken token)
        {
            while (!token.IsCancellationRequested)
            {
                foreach (var trigger in Triggers)
                {
                    foreach (var issue in GitHubAPI.GetIssues())
                    {
                        if (trigger.Condition(issue))
                        {
                            trigger.Action(issue);
                        }
                    }
                }
                Thread.Sleep(MinimumInteractionInterval);
            }
        }


        /// <summary>
        /// <para>
        /// Starts the cancellation token.
        /// </para>
        /// <para></para>
        /// </summary>
        /// <param name="cancellationToken">
        /// <para>The cancellation token.</para>
        /// <para></para>
        /// </param>
        public void Start(CancellationToken cancellationToken) =>
         ↪  ProcessIssues(cancellationToken);
    }
}
```

1.3 ./csharp/Platform.Bot/Triggers/Activity.cs

```
using System.Collections.Generic;

namespace Bot
{
    internal class Activity
    {
        public string Url { get; set; }

        public List<string> Repositories { get; set; }
    }
}
```

1.4 ./csharp/Platform.Bot/Triggers/HelloWorldTrigger.cs

```
using Interfaces;
using Octokit;
using Storage.Local;
using Storage.Remote.GitHub;

namespace csharp
{
    /// <summary>
    /// <para>
    /// Represents the hello world trigger.
    /// </para>
    /// <para></para>
    /// </summary>
    /// <seealso cref="ITrigger{Issue}"/>
    internal class HelloWorldTrigger : ITrigger<Issue>
    {
        /// <summary>
        /// <para>
        /// The git hub api.
        /// </para>
        /// <para></para>
        /// </summary>
        private readonly GitHubStorage gitHubAPI;

        /// <summary>
        /// <para>
        /// The file storage.
        /// </para>
        /// <para></para>
        /// </summary>
        private readonly FileStorage fileStorage;

        /// <summary>
        /// <para>
        /// The file set name.
        /// </para>
        /// <para></para>
        /// </summary>
        private readonly string fileSetName;

        /// <summary>
        /// <para>
        /// Initializes a new <see cref="HelloWorldTrigger"/> instance.
        /// </para>
        /// <para></para>
        /// </summary>
        /// <param name="gitHubAPI">
        /// <para>A git hub api.</para>
        /// <para></para>
        /// </param>
```

```csharp
        /// <param name="fileStorage">
        /// <para>A file storage.</para>
        /// <para></para>
        /// </param>
        /// <param name="fileSetName">
        /// <para>A file set name.</para>
        /// <para></para>
        /// </param>
        public HelloWorldTrigger(GitHubStorage gitHubAPI, FileStorage fileStorage, string
         ↪  fileSetName)
        {
            this.gitHubAPI = gitHubAPI;
            this.fileStorage = fileStorage;
            this.fileSetName = fileSetName;
        }


        /// <summary>
        /// <para>
        /// Actions the obj.
        /// </para>
        /// <para></para>
        /// </summary>
        /// <param name="obj">
        /// <para>The obj.</para>
        /// <para></para>
        /// </param>

        public void Action(Issue obj)
        {
            foreach (var file in fileStorage.GetFilesFromSet(fileSetName))
            {
                gitHubAPI.CreateOrUpdateFile(obj.Repository.Name, obj.Repository.DefaultBranch,
                 ↪  file);
            }
            gitHubAPI.CloseIssue(obj);
        }


        /// <summary>
        /// <para>
        /// Determines whether this instance condition.
        /// </para>
        /// <para></para>
        /// </summary>
        /// <param name="obj">
        /// <para>The obj.</para>
        /// <para></para>
        /// </param>
        /// <returns>
        /// <para>The bool</para>
        /// <para></para>
        /// </returns>
        public bool Condition(Issue obj) => obj.Title.ToLower() == "hello world";
    }
}
```

1.5 ./csharp/Platform.Bot/Triggers/LastCommitActivityTrigger.cs

```csharp
using Interfaces;
using Octokit;
using Platform.Communication.Protocol.Lino;
using Storage.Remote.GitHub;
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace Bot
{
    internal class LastCommitActivityTrigger : ITrigger<Issue>
    {
        private readonly GitHubStorage Storage;

        private readonly Parser Parser = new();

        public LastCommitActivityTrigger(GitHubStorage storage) => Storage = storage;

        public bool Condition(Issue issue) => issue.Title.ToLower() == "last 3 months commit
         ↪  activity";

        public void Action(Issue issue)
        {
            var issueService = Storage.Client.Issue;
            var owner = issue.Repository.Owner.Login;
            var users = GetActivities(GetIgnoredRepositories(Parser.Parse(issue.Body)), owner);
            StringBuilder sb = new();
            sb.Append("```");
            foreach (var user in users)
            {
                sb.AppendLine($"{Environment.NewLine}" + user.Url.Replace("api.",
                ↪  "").Replace("users/", ""));
                foreach (var repo in user.Repositories)
                {
                    sb.AppendLine(repo.Replace("api.", "").Replace("repos/", ""));
                }
            }
            Console.WriteLine(sb.Append("```").ToString());
            issueService.Comment.Create(owner, issue.Repository.Name, issue.Number,
            ↪  sb.Append("```").ToString());
            Storage.CloseIssue(issue);
        }

        public HashSet<string> GetIgnoredRepositories(IList<Link> links)
        {
            HashSet<string> ignoredRepos = new() { };
            foreach (var link in links)
            {
                var values = link.Values;
                if (values != null && values.Count == 3 && string.Equals(values.First().Id,
                ↪  "ignore", StringComparison.OrdinalIgnoreCase) &&
                ↪  string.Equals(values.Last().Id.Trim('.'), "repository",
                ↪  StringComparison.OrdinalIgnoreCase))
                {
                    ignoredRepos.Add(values[1].Id);
                }
            }
            return ignoredRepos;
        }

        public HashSet<Activity> GetActivities(HashSet<string> ignoredRepositories, string owner)
        {
            HashSet<Activity> activeUsers = new();
            foreach (var repository in Storage.Client.Repository.GetAllForOrg(owner).Result)
            {
                if (ignoredRepositories.Contains(repository.Name))
                {
                    continue;
                }
                foreach (var commit in Storage.GetCommits(repository.Owner.Login,
                ↪  repository.Name, DateTime.Today.AddMonths(-3)))
                {
                    if (!activeUsers.Any(x => x.Url == commit.Author.Url))
                    {
                        activeUsers.Add(new Activity() { Url = commit.Author.Url, Repositories =
                        ↪  new List<string> { repository.Url } });
                    }
                    else
                    {
                        if (!activeUsers.Any(x => x.Repositories.Any(y => y == repository.Url)
                        ↪  == true))
                        {
                            activeUsers.FirstOrDefault(x => x.Url ==
                            ↪  commit.Author.Url).Repositories.Add(repository.Url);
                        }
                    }
                }
            }
            return activeUsers;
        }
    }
}
```

1.6 ./csharp/Platform.Bot/Triggers/OrganizationLastMonthActivityTrigger.cs

```csharp
using Interfaces;
using Octokit;
using Platform.Communication.Protocol.Lino;
using Storage.Remote.GitHub;
using System;
```

```
using System.Collections.Generic;
using System.Linq;

namespace Platform.Bot
{
    /// <summary>
    /// <para>
    /// Represents the organization last month activity trigger.
    /// </para>
    /// <para></para>
    /// </summary>
    /// <seealso cref="ITrigger{Issue}"/>
    internal class OrganizationLastMonthActivityTrigger : ITrigger<Issue>
    {
        /// <summary>
        /// <para>
        /// The storage.
        /// </para>
        /// <para></para>
        /// </summary>
        private readonly GitHubStorage Storage;

        /// <summary>
        /// <para>
        /// The parser.
        /// </para>
        /// <para></para>
        /// </summary>
        private readonly Parser Parser = new();

        /// <summary>
        /// <para>
        /// Initializes a new <see cref="OrganizationLastMonthActivityTrigger"/> instance.
        /// </para>
        /// <para></para>
        /// </summary>
        /// <param name="storage">
        /// <para>A storage.</para>
        /// <para></para>
        /// </param>
        public OrganizationLastMonthActivityTrigger(GitHubStorage storage) => Storage = storage;

        /// <summary>
        /// <para>
        /// Determines whether this instance condition.
        /// </para>
        /// <para></para>
        /// </summary>
        /// <param name="issue">
        /// <para>The issue.</para>
        /// <para></para>
        /// </param>
        /// <returns>
        /// <para>The bool</para>
        /// <para></para>
        /// </returns>
        public bool Condition(Issue issue) => issue.Title.ToLower() == "organization last month
         ↪  activity";

        /// <summary>
        /// <para>
        /// Actions the issue.
        /// </para>
        /// <para></para>
        /// </summary>
        /// <param name="issue">
        /// <para>The issue.</para>
        /// <para></para>
        /// </param>
        public void Action(Issue issue)
        {
            var issueService = Storage.Client.Issue;
            var owner = issue.Repository.Owner.Login;
            var activeUsersString = string.Join("\n",
             ↪  GetActiveUsers(GetIgnoredRepositories(Parser.Parse(issue.Body)), owner));
            issueService.Comment.Create(owner, issue.Repository.Name, issue.Number,
             ↪  activeUsersString);
            Storage.CloseIssue(issue);
        }

        /// <summary>
        /// <para>
        /// Gets the ignored repositories using the specified links.
        /// </para>
        /// <para></para>
        /// </summary>
        /// <param name="links">
        /// <para>The links.</para>
        /// <para></para>
        /// </param>
        /// <returns>
        /// <para>The ignored repos.</para>
        /// <para></para>
        /// </returns>
        public HashSet<string> GetIgnoredRepositories(IList<Link> links)
        {
            HashSet<string> ignoredRepos = new() { };
            foreach (var link in links)
            {
                var values = link.Values;
                if (values != null && values.Count == 3 && string.Equals(values.First().Id,
                ↪   "ignore", StringComparison.OrdinalIgnoreCase) &&
                ↪   string.Equals(values.Last().Id.Trim('.'), "repository",
                ↪   StringComparison.OrdinalIgnoreCase))
                {
                    ignoredRepos.Add(values[1].Id);
                }
            }
            return ignoredRepos;
        }

        /// <summary>
        /// <para>
        /// Gets the active users using the specified ignored repositories.
        /// </para>
        /// <para></para>
        /// </summary>
        /// <param name="ignoredRepositories">
        /// <para>The ignored repositories.</para>
        /// <para></para>
        /// </param>
        /// <param name="owner">
        /// <para>The owner.</para>
        /// <para></para>
        /// </param>
        /// <returns>
        /// <para>The active users.</para>
        /// <para></para>
        /// </returns>
        public HashSet<string> GetActiveUsers(HashSet<string> ignoredRepositories, string owner)
        {
            HashSet<string> activeUsers = new();
            foreach (var repository in Storage.Client.Repository.GetAllForOrg(owner).Result)
            {
                if (ignoredRepositories.Contains(repository.Name))
                {
                    continue;
                }
                foreach (var commit in Storage.GetCommits(repository.Owner.Login,
                ↪   repository.Name, date))
                {

                    activeUsers.Add(commit.Author.Login);

                }
                foreach (var pullRequest in Storage.GetPullRequests(repository.Owner.Login,
                ↪   repository.Name))
                {
                    foreach (var reviewer in pullRequest.RequestedReviewers)
                    {
                        if (pullRequest.CreatedAt < date || pullRequest.UpdatedAt < date ||
                        ↪   pullRequest.ClosedAt < date || pullRequest.MergedAt < date)
                        {
                            activeUsers.Add(reviewer.Login);
                        }
                    }
                }
                foreach (var createdIssue in Storage.GetIssues(repository.Owner.Login, repository.Name))
                {
                    activeUsers.Add(createdIssue.User.Login);
                }
            }
            return activeUsers;
        }
    }
}
```

Index