LinksPlatform's Platform.Bot Class Library

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

```csharp
1   using Interfaces;
2   using Octokit;
3   using Platform.Exceptions;
4   using Platform.IO;
5   using Storage.Local;
6   using Storage.Remote.GitHub;
7   using System;
8   using System.Collections.Generic;
9   using System.Threading;
10  using Platform.Bot.Trackers;
11  using Platform.Bot.Triggers;
12
13  namespace Platform.Bot
14  {
15      /// <summary>
16      /// <para>
17      /// Represents the program.
18      /// </para>
19      /// <para></para>
20      /// </summary>
21      internal class Program
22      {
23          private static void Main(string[] args)
24          {
25              using var cancellation = new ConsoleCancellation();
26              var argumentIndex = 0;
27              var username = ConsoleHelpers.GetOrReadArgument(argumentIndex++, "Username", args);
28              var token = ConsoleHelpers.GetOrReadArgument(argumentIndex++, "Token", args);
29              var appName = ConsoleHelpers.GetOrReadArgument(argumentIndex++, "App Name", args);
30              var databaseFileName = ConsoleHelpers.GetOrReadArgument(argumentIndex++, "Database
    ↪   file name", args);
31              var fileSetName = ConsoleHelpers.GetOrReadArgument(argumentIndex++, "File set name",
    ↪   args);
32              var minimumInteractionIntervalStringInputInSeconds =
    ↪   ConsoleHelpers.GetOrReadArgument(argumentIndex, "Minimum interaction interval in
    ↪   seconds", args);
33              var minimumInteractionInterval = TimeSpan.FromSeconds(Int32.Parse(minimumInteraction
    ↪   IntervalStringInputInSeconds));
34              var dbContext = new FileStorage(databaseFileName);
35              Console.WriteLine($"Bot has been started. {Environment.NewLine}Press CTRL+C to
    ↪   close");
36              try
37              {
38                  while (true)
39                  {
40                      var api = new GitHubStorage(username, token, appName);
41                      new IssueTracker(new List<ITrigger<Issue>>
42                      {
43                          new HelloWorldTrigger(api, dbContext, fileSetName),
44                          new OrganizationLastMonthActivityTrigger(api),
45                          new LastCommitActivityTrigger(api),
46                          new ProtectMainBranchTrigger(api),
47                      }, api).Start(cancellation.Token);
48                      new PullRequestTracker(new List<ITrigger<PullRequest>> { new
    ↪   MergeDependabotBumpsTrigger(api) }, api).Start(cancellation.Token);
49                      Thread.Sleep(minimumInteractionInterval);
50                  }
51              }
52              catch (Exception ex)
53              {
54                  Console.WriteLine(ex.ToStringWithAllInnerExceptions());
55              }
56          }
57      }
58  }
```

1.2 ./csharp/Platform.Bot/Trackers/IssueTracker.cs

```csharp
1   using Interfaces;
2   using Octokit;
3   using Storage.Remote.GitHub;
4   using System.Collections.Generic;
5   using System.Threading;
6
7   namespace Platform.Bot.Trackers
8   {
9       /// <summary>
10      /// <para>
```

```csharp
     /// Represents the programmer role.
     /// </para>
     /// <para></para>
     /// </summary>
     public class IssueTracker : ITracker<Issue>
     {
         /// <summary>
         /// <para>
         /// The git hub api.
         /// </para>
         /// <para></para>
         /// </summary>
         public GitHubStorage Storage { get; }

         /// <summary>
         /// <para>
         /// The triggers.
         /// </para>
         /// <para></para>
         /// </summary>
         public List<ITrigger<Issue>> Triggers { get; }

         /// <summary>
         /// <para>
         /// Initializes a new <see cref="IssueTracker"/> 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 IssueTracker(List<ITrigger<Issue>> triggers, GitHubStorage gitHubApi)
         {
             Storage = gitHubApi;
             Triggers = triggers;
         }

         /// <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)
         {
             foreach (var trigger in Triggers)
             {
                 foreach (var issue in Storage.GetIssues())
                 {
                     if (cancellationToken.IsCancellationRequested)
                     {
                         return;
                     }
                     if (trigger.Condition(issue))
                     {
                         trigger.Action(issue);
                     }
                 }
             }
         }
     }
}
```

1.3 ./csharp/Platform.Bot/Trackers/PullRequestTracker.cs

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

```csharp
namespace Platform.Bot.Trackers
{
    using TContext = PullRequest;
    /// <summary>
    /// <para>
    /// Represents the programmer role.
    /// </para>
    /// <para></para>
    /// </summary>
    public class PullRequestTracker : ITracker<TContext>
    {
        /// <summary>
        /// <para>
        /// The git hub api.
        /// </para>
        /// <para></para>
        /// </summary>
        public GitHubStorage Storage { get; }

        /// <summary>
        /// <para>
        /// The triggers.
        /// </para>
        /// <para></para>
        /// </summary>
        public List<ITrigger<TContext>> Triggers { get; }

        /// <summary>
        /// <para>
        /// Initializes a new <see cref="IssueTracker"/> instance.
        /// </para>
        /// <para></para>
        /// </summary>
        /// <param name="triggers">
        /// <para>A triggers.</para>
        /// <para></para>
        /// </param>
        /// <param name="storage">
        /// <para>A git hub api.</para>
        /// <para></para>
        /// </param>
        public PullRequestTracker(List<ITrigger<TContext>> triggers, GitHubStorage storage)
        {
            Storage = storage;
            Triggers = triggers;
        }

        /// <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)
        {
            foreach (var trigger in Triggers)
            {
                foreach (var repository in
                ↪   Storage.Client.Repository.GetAllForOrg("linksplatform").AwaitResult())
                {
                    foreach (var pullRequest in Storage.Client.PullRequest.GetAllForRepository(r
                    ↪   epository.Id).AwaitResult())
                    {
                        if (cancellationToken.IsCancellationRequested)
                        {
                            return;
                        }
                        if (trigger.Condition(pullRequest))
                        {
                            trigger.Action(pullRequest);
                        }
                    }
                }
            }
```

```
83                    }
84                }
85            }
86    }
```

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

```csharp
1    using System.Collections.Generic;
2
3    namespace Platform.Bot.Triggers
4    {
5        internal class Activity
6        {
7            public string Url { get; set; }
8
9            public List<string> Repositories { get; set; }
10        }
11    }
```

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

```csharp
1    using Interfaces;
2    using Octokit;
3    using Storage.Local;
4    using Storage.Remote.GitHub;
5
6    namespace Platform.Bot.Triggers
7    {
8        using TContext = Issue;
9        /// <summary>
10       /// <para>
11       /// Represents the hello world trigger.
12       /// </para>
13       /// <para></para>
14       /// </summary>
15       /// <seealso cref="ITrigger{TContext}"/>
16       internal class HelloWorldTrigger : ITrigger<TContext>
17       {
18           private readonly GitHubStorage _storage;
19           private readonly FileStorage _fileStorage;
20           private readonly string _fileSetName;
21
22           /// <summary>
23           /// <para>
24           /// Initializes a new <see cref="HelloWorldTrigger"/> instance.
25           /// </para>
26           /// <para></para>
27           /// </summary>
28           /// <param name="storage">
29           /// <para>A git hub api.</para>
30           /// <para></para>
31           /// </param>
32           /// <param name="fileStorage">
33           /// <para>A file storage.</para>
34           /// <para></para>
35           /// </param>
36           /// <param name="fileSetName">
37           /// <para>A file set name.</para>
38           /// <para></para>
39           /// </param>
40           public HelloWorldTrigger(GitHubStorage storage, FileStorage fileStorage, string
              ↪   fileSetName)
41           {
42               this._storage = storage;
43               this._fileStorage = fileStorage;
44               this._fileSetName = fileSetName;
45           }
46
47
48           /// <summary>
49           /// <para>
50           /// Actions the context.
51           /// </para>
52           /// <para></para>
53           /// </summary>
54           /// <param name="context">
55           /// <para>The context.</para>
56           /// <para></para>
57           /// </param>
58
59           public void Action(TContext context)
```

```csharp
60              {
61                  foreach (var file in _fileStorage.GetFilesFromSet(_fileSetName))
62                  {
63                      _storage.CreateOrUpdateFile(context.Repository.Name,
   ↪    context.Repository.DefaultBranch, file);
64                  }
65                  _storage.CloseIssue(context);
66              }
67
68
69          /// <summary>
70          /// <para>
71          /// Determines whether this instance condition.
72          /// </para>
73          /// <para></para>
74          /// </summary>
75          /// <param name="context">
76          /// <para>The context.</para>
77          /// <para></para>
78          /// </param>
79          /// <returns>
80          /// <para>The bool</para>
81          /// <para></para>
82          /// </returns>
83          public bool Condition(TContext context) => context.Title.ToLower() == "hello world";
84          }
85  }
```

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

```csharp
1   using Interfaces;
2   using Octokit;
3   using Platform.Communication.Protocol.Lino;
4   using Storage.Remote.GitHub;
5   using System;
6   using System.Collections.Generic;
7   using System.Linq;
8   using System.Text;
9
10  namespace Platform.Bot.Triggers
11  {
12      using TContext = Issue;
13      internal class LastCommitActivityTrigger : ITrigger<TContext>
14      {
15          private readonly GitHubStorage _storage;
16
17          private readonly Parser _parser = new();
18
19          public LastCommitActivityTrigger(GitHubStorage storage) => _storage = storage;
20
21          public bool Condition(TContext context) => context.Title.ToLower() == "last 3 months
   ↪    commit activity";
22
23          public void Action(TContext context)
24          {
25              var issueService = _storage.Client.Issue;
26              var owner = context.Repository.Owner.Login;
27              var ignoredRepositories =
28                  context.Body != null ? GetIgnoredRepositories(_parser.Parse(context.Body)) :
   ↪    default;
29              var users = GetActivities(owner, ignoredRepositories);
30              StringBuilder sb = new();
31              foreach (var user in users)
32              {
33                  sb.AppendLine($"- **{user.Url.Replace("api.", "").Replace("users/", "")}**");
34                  // Break line
35                  sb.AppendLine("------------------");
36              }
37              var result = sb.ToString();
38              var comment = issueService.Comment.Create(owner, context.Repository.Name,
   ↪    context.Number, result);
39              comment.Wait();
40              Console.WriteLine($"Last commit activity comment is added:
   ↪    {comment.Result.HtmlUrl}");
41              _storage.CloseIssue(context);
42          }
43
44          public HashSet<string> GetIgnoredRepositories(IList<Link> links)
45          {
46              HashSet<string> ignoredRepos = new() { };
```

```csharp
                    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(string owner, HashSet<string> ignoredRepositories
                ↪    = default)
                {
                    HashSet<Activity> activeUsers = new();
                    foreach (var repository in _storage.Client.Repository.GetAllForOrg(owner).Result)
                    {
                        if (ignoredRepositories?.Contains(repository.Name) ?? false)
                        {
                            continue;
                        }
                        foreach (var commit in _storage.GetCommits(repository.Owner.Login,
                        ↪    repository.Name, DateTime.Today.AddMonths(-3)))
                        {
                            if (!_storage.Client.Organization.Member.CheckMember(owner,
                            ↪    commit.Author.Login).Result)
                            {
                                continue;
                            }
                            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;
            }
        }
    }
}
```

1.7 ./csharp/Platform.Bot/Triggers/MergeDependabotBumpsTrigger.cs

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

namespace Platform.Bot.Triggers
{
    using TContext = PullRequest;
    public class MergeDependabotBumpsTrigger : ITrigger<TContext>
    {
        private const int DependabotId = 49699333;
        private readonly GitHubStorage _storage;
        public MergeDependabotBumpsTrigger(GitHubStorage storage)
        {
            _storage = storage;
        }

        public bool Condition(TContext context)
        {
            var isDependabotAuthor = DependabotId == context.User.Id;
            var isTestAndDeployCompleted = false;
            var repositoryId = context.Base.Repository.Id;
            var checks = _storage.Client.Check.Run.GetAllForReference(repositoryId,
            ↪    context.Head.Sha).AwaitResult();
```

```csharp
25              foreach (var checkRun in checks.CheckRuns)
26              {
27                  if (checkRun.Name == "testAndDeploy" && checkRun.Status.Value ==
   ↪   CheckStatus.Completed)
28                  {
29                      isTestAndDeployCompleted = true;
30                  }
31              }
32              return isDependabotAuthor && isTestAndDeployCompleted;
33          }
34
35          public void Action(TContext context)
36          {
37              var repositoryId = context.Base.Repository.Id;
38              var prMerge = _storage.Client.PullRequest.Merge(repositoryId, context.Number, new
   ↪   MergePullRequest()).AwaitResult();
39              Console.WriteLine($"{context.HtmlUrl} is {(prMerge.Merged ? "successfully":"not
   ↪   successfully")} merged.");
40          }
41      }
42  }
```

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

```csharp
1   using System;
2   using System.Collections.Generic;
3   using System.Linq;
4   using Interfaces;
5   using Octokit;
6   using Platform.Communication.Protocol.Lino;
7   using Storage.Remote.GitHub;
8
9   namespace Platform.Bot.Triggers
10  {
11      using TContext = Issue;
12      /// <summary>
13      /// <para>
14      /// Represents the organization last month activity trigger.
15      /// </para>
16      /// <para></para>
17      /// </summary>
18      /// <seealso cref="ITrigger{Issue}"/>
19      internal class OrganizationLastMonthActivityTrigger : ITrigger<TContext>
20      {
21          private readonly GitHubStorage _storage;
22          private readonly Parser _parser = new();
23
24          /// <summary>
25          /// <para>
26          /// Initializes a new <see cref="OrganizationLastMonthActivityTrigger"/> instance.
27          /// </para>
28          /// <para></para>
29          /// </summary>
30          /// <param name="storage">
31          /// <para>A storage.</para>
32          /// <para></para>
33          /// </param>
34          public OrganizationLastMonthActivityTrigger(GitHubStorage storage) => _storage = storage;
35
36          /// <summary>
37          /// <para>
38          /// Determines whether this instance condition.
39          /// </para>
40          /// <para></para>
41          /// </summary>
42          /// <param name="context">
43          /// <para>The context.</para>
44          /// <para></para>
45          /// </param>
46          /// <returns>
47          /// <para>The bool</para>
48          /// <para></para>
49          /// </returns>
50          public bool Condition(TContext context) => context.Title.ToLower() == "organization last
   ↪   month activity";
51
52          /// <summary>
53          /// <para>
54          /// Actions the context.
55          /// </para>
```

```csharp
        /// <para></para>
        /// </summary>
        /// <param name="context">
        /// <para>The context.</para>
        /// <para></para>
        /// </param>
        public void Action(TContext context)
        {
            var issueService = _storage.Client.Issue;
            var owner = context.Repository.Owner.Login;
            var activeUsersString = string.Join("\n",
            ↪  GetActiveUsers(GetIgnoredRepositories(_parser.Parse(context.Body)), owner));
            issueService.Comment.Create(owner, context.Repository.Name, context.Number,
            ↪  activeUsersString);
            _storage.CloseIssue(context);
        }

        /// <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();
            var date = DateTime.Now.AddMonths(-1);
            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))
```

```
128                    {
129
130                        activeUsers.Add(commit.Author.Login);
131
132                    }
133                    foreach (var pullRequest in _storage.GetPullRequests(repository.Owner.Login,
        ↪   repository.Name))
134                    {
135                        foreach (var reviewer in pullRequest.RequestedReviewers)
136                        {
137                            if (pullRequest.CreatedAt < date || pullRequest.UpdatedAt < date ||
            ↪   pullRequest.ClosedAt < date || pullRequest.MergedAt < date)
138                            {
139                                activeUsers.Add(reviewer.Login);
140                            }
141                        }
142                    }
143                    foreach (var createdIssue in _storage.GetIssues(repository.Owner.Login,
        ↪   repository.Name))
144                    {
145                        activeUsers.Add(createdIssue.User.Login);
146                    }
147                }
148                return activeUsers;
149            }
150        }
151    }
```

1.9 ./csharp/Platform.Bot/Triggers/ProtectMainBranchTrigger.cs

```
1   using System.Collections.Generic;
2   using System.Linq;
3   using System.Text;
4   using Interfaces;
5   using Octokit;
6   using Storage.Remote.GitHub;
7
8   namespace Platform.Bot.Triggers
9   {
10      using TContext = Issue;
11      public class ProtectMainBranchTrigger : ITrigger<TContext>
12      {
13          private readonly GitHubStorage _storage;
14          public ProtectMainBranchTrigger(GitHubStorage storage) => _storage = storage;
15          public bool Condition(TContext context) => context.Title.ToLower() == "protect default
        ↪   branch in all organization's repositories";
16
17          public void Action(TContext context)
18          {
19              var repositories =
            ↪   _storage.Client.Repository.GetAllForOrg(context.Repository.Owner.Login).Result;
20              var results = UpdateRepositoriesDefaultBranchProtection(repositories);
21              StringBuilder failedRepositoriesComment = new(repositories.Count *
            ↪   repositories[0].Name.Length);
22              foreach (var result in results.Where(result => !result.Value))
23              {
24                  failedRepositoriesComment.AppendLine($"- [ ] {result.Key}");
25              }
26              if (failedRepositoriesComment.Length != 0)
27              {
28                  failedRepositoriesComment.AppendLine(
29                      "TODO: Fix default branch protection of these repositories. Failed
                    ↪   repositories:");
30                  _storage.Client.Issue.Comment.Create(context.Repository.Id, context.Number,
                ↪   failedRepositoriesComment.ToString());
31              }
32              else
33              {
34                  _storage.Client.Issue.Comment.Create(context.Repository.Id, context.Number,
                ↪   "Success. All repositories default branch protection is updated.");
35                  _storage.CloseIssue(context);
36              }
37          }
38
39          public Dictionary<string, bool>
            ↪   UpdateRepositoriesDefaultBranchProtection(IReadOnlyList<Repository> repositories)
40          {
41              Dictionary<string, bool> result = new(repositories.Count);
42              foreach (var repository in repositories)
```

```csharp
            {
                if (repository.Private)
                {
                    continue;
                }
                var update = new BranchProtectionSettingsUpdate(new
                ↪  BranchProtectionPushRestrictionsUpdate());
                var request =
                    _storage.Client.Repository.Branch.UpdateBranchProtection(repository.Id,
                        repository.DefaultBranch, update);
                request.Wait();
                result.Add(repository.Name, request.IsCompletedSuccessfully);
            }
            return result;
        }
    }
}
```

Index