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

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

namespace Platform.Bot
{
    /// <summary>
    /// <para>
    /// Represents the program.
    /// </para>
    /// <para></para>
    /// </summary>
    internal class Program
    {
        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 IssueTracker(
                    new List<ITrigger<Issue>> {
                        new HelloWorldTrigger(api, dbContext, fileSetName),
                        new OrganizationLastMonthActivityTrigger(api),
                        new LastCommitActivityTrigger(api),
                        new ProtectMainBranchTrigger(api),
                    },
                    api
                ).Start(cancellation.Token);
                new PullRequestTracker(new List<ITrigger<PullRequest>> { new MergeDependabotBumpsTrigger(api) }, api).Start(cancellation.Token);
            }
            catch (Exception ex)
            {
                Console.WriteLine(ex.ToStringWithAllInnerExceptions());
            }
        }
    }
}
```

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

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

namespace Platform.Bot.Trackers
{
    /// <summary>
    /// <para>
    /// Represents the programmer role.
    /// </para>
    /// <para></para>
    /// </summary>
    public class IssueTracker : ITracker<Issue>
    {
        /// <summary>
```

```
19          /// <para>
20          /// The git hub api.
21          /// </para>
22          /// <para></para>
23          /// </summary>
24          public GitHubStorage GitHubApi { get; }
25
26          /// <summary>
27          /// <para>
28          /// The minimum interaction interval.
29          /// </para>
30          /// <para></para>
31          /// </summary>
32          public TimeSpan MinimumInteractionInterval { get; }
33
34          /// <summary>
35          /// <para>
36          /// The triggers.
37          /// </para>
38          /// <para></para>
39          /// </summary>
40          public List<ITrigger<Issue>> Triggers { get; }
41
42          /// <summary>
43          /// <para>
44          /// Initializes a new <see cref="IssueTracker"/> instance.
45          /// </para>
46          /// <para></para>
47          /// </summary>
48          /// <param name="triggers">
49          /// <para>A triggers.</para>
50          /// <para></para>
51          /// </param>
52          /// <param name="gitHubAPI">
53          /// <para>A git hub api.</para>
54          /// <para></para>
55          /// </param>
56          public IssueTracker(List<ITrigger<Issue>> triggers, GitHubStorage gitHubApi)
57          {
58              GitHubApi = gitHubApi;
59              Triggers = triggers;
60              MinimumInteractionInterval = gitHubApi.MinimumInteractionInterval;
61          }
62
63
64          /// <summary>
65          /// <para>
66          /// Starts the cancellation token.
67          /// </para>
68          /// <para></para>
69          /// </summary>
70          /// <param name="cancellationToken">
71          /// <para>The cancellation token.</para>
72          /// <para></para>
73          /// </param>
74          public void Start(CancellationToken cancellationToken)
75          {
76              while (!cancellationToken.IsCancellationRequested)
77              {
78                  foreach (var trigger in Triggers)
79                  {
80                      foreach (var issue in GitHubApi.GetIssues())
81                      {
82                          if (trigger.Condition(issue))
83                          {
84                              trigger.Action(issue);
85                          }
86                      }
87                  }
88                  Thread.Sleep(MinimumInteractionInterval);
89              }
90          }
91      }
92  }
```

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

```
1   using Interfaces;
2   using Octokit;
3   using Storage.Remote.GitHub;
```

```
using System;
using System.Collections.Generic;
using System.Threading;
using Platform.Threading;

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 GitHubApi { get; }


        /// <summary>
        /// <para>
        /// The minimum interaction interval.
        /// </para>
        /// <para></para>
        /// </summary>
        public TimeSpan MinimumInteractionInterval { 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="gitHubApi">
        /// <para>A git hub api.</para>
        /// <para></para>
        /// </param>
        public PullRequestTracker(List<ITrigger<TContext>> triggers, GitHubStorage gitHubApi)
        {
            GitHubApi = gitHubApi;
            Triggers = triggers;
            MinimumInteractionInterval = gitHubApi.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)
        {
            while (!cancellationToken.IsCancellationRequested)
            {
                foreach (var trigger in Triggers)
                {
```

```
                    foreach (var repository in GitHubApi.Client.Repository.GetAllForOrg("linksplatform").AwaitResult())
                    {
                        foreach (var pullRequest in GitHubApi.Client.PullRequest.GetAllForRepository(repository.Id).AwaitResult())
                        {
                            if (trigger.Condition(pullRequest))
                            {
                                trigger.Action(pullRequest);
                            }
                        }
                    }
                }
                Thread.Sleep(MinimumInteractionInterval);
            }
        }
    }
}
```

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

```
using System.Collections.Generic;

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

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

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

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

namespace Platform.Bot.Triggers
{
    using TContext = Issue;
    /// <summary>
    /// <para>
    /// Represents the hello world trigger.
    /// </para>
    /// <para></para>
    /// </summary>
    /// <seealso cref="ITrigger{TContext}"/>
    internal class HelloWorldTrigger : ITrigger<TContext>
    {
        private readonly GitHubStorage gitHubApi;
        private readonly FileStorage fileStorage;
        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>
        /// <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 context.
        /// </para>
        /// <para></para>
        /// </summary>
        /// <param name="context">
        /// <para>The context.</para>
        /// <para></para>
        /// </param>

        public void Action(TContext context)
        {
            foreach (var file in fileStorage.GetFilesFromSet(fileSetName))
            {
                gitHubApi.CreateOrUpdateFile(context.Repository.Name,
                ↪  context.Repository.DefaultBranch, file);
            }
            gitHubApi.CloseIssue(context);
        }


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

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

```
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 Platform.Bot.Triggers
{
    using TContext = Issue;
    internal class LastCommitActivityTrigger : ITrigger<TContext>
    {
        private readonly GitHubStorage Storage;

        private readonly Parser Parser = new();

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

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

        public void Action(TContext context)
        {
            var issueService = Storage.Client.Issue;
            var owner = context.Repository.Owner.Login;
            var ignoredRepositories =
                context.Body != null ? GetIgnoredRepositories(Parser.Parse(context.Body)) :
                ↪  default;
            var users = GetActivities(owner, ignoredRepositories);
            StringBuilder sb = new();
            foreach (var user in users)
            {
                sb.AppendLine($"- **{user.Url.Replace("api.", "").Replace("users/", "")}**");
                // Break line
                sb.AppendLine("------------------");
```

```
            }
            var result = sb.ToString();
            var comment = issueService.Comment.Create(owner, context.Repository.Name,
            ↪  context.Number, result);
            comment.Wait();
            Console.WriteLine($"Last commit activity comment is added:
            ↪  {comment.Result.HtmlUrl}");
            Storage.CloseIssue(context);
        }

        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(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

```
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 readonly 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();
            foreach (var checkRun in checks.CheckRuns)
            {
                if (checkRun.Name == "testAndDeploy" && checkRun.Status.Value ==
                ↪  CheckStatus.Completed)
                {
                    isTestAndDeployCompleted = true;
                }
            }
            return isDependabotAuthor && isTestAndDeployCompleted;
        }

        public void Action(TContext context)
        {
            var repositoryId = context.Base.Repository.Id;
            var prMerge = Storage.Client.PullRequest.Merge(repositoryId, context.Number, new
            ↪  MergePullRequest()).AwaitResult();
            Console.WriteLine($"{context.HtmlUrl} is {(prMerge.Merged ? "successfully":"not
            ↪  successfully")} merged.");
        }
    }
}
```

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

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

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

```
        /// <para>The context.</para>
        /// <para></para>
        /// </param>
        /// <returns>
        /// <para>The bool</para>
        /// <para></para>
        /// </returns>
        public bool Condition(TContext context) => context.Title.ToLower() == "organization last
        ↪  month activity";

        /// <summary>
        /// <para>
        /// Actions the context.
        /// </para>
        /// <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))
                {

                    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;
        }
    }
}
```

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

```
using System.Collections.Generic;
using System.Linq;
using System.Text;
using Interfaces;
using Octokit;
using Storage.Remote.GitHub;

namespace Platform.Bot.Triggers
{
    using TContext = Issue;
    public class ProtectMainBranchTrigger : ITrigger<TContext>
    {
        private readonly GitHubStorage Storage;
        public ProtectMainBranchTrigger(GitHubStorage storage) => Storage = storage;
        public bool Condition(TContext context) => context.Title.ToLower() == "protect default
        ↪  branch in all organization's repositories";

        public void Action(TContext context)
        {
            var repositories =
            ↪  Storage.Client.Repository.GetAllForOrg(context.Repository.Owner.Login).Result;
            var results = UpdateRepositoriesDefaultBranchProtection(repositories);
            StringBuilder failedRepositoriesComment = new(repositories.Count *
            ↪  repositories[0].Name.Length);
            foreach (var result in results.Where(result => !result.Value))
            {
                failedRepositoriesComment.AppendLine($"- [ ] {result.Key}");
            }
            if (failedRepositoriesComment.Length != 0)
            {
                failedRepositoriesComment.AppendLine(
                    "TODO: Fix default branch protection of these repositories. Failed
                    ↪  repositories:");
                Storage.Client.Issue.Comment.Create(context.Repository.Id, context.Number,
                ↪  failedRepositoriesComment.ToString());
```

```
            }
            else
            {
                Storage.Client.Issue.Comment.Create(context.Repository.Id, context.Number,
                ↪  "Success. All repositories default branch protection is updated.");
                Storage.CloseIssue(context);
            }
        }

        public Dictionary<string, bool>
        ↪  UpdateRepositoriesDefaultBranchProtection(IReadOnlyList<Repository> repositories)
        {
            Dictionary<string, bool> result = new(repositories.Count);
            foreach (var repository in repositories)
            {
                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