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;

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),
                        new ProtectMainBranchTrigger(api)
                    },
                    api
                ).Start(cancellation.Token);
            }
            catch (Exception ex)
            {
                Console.WriteLine(ex.ToStringWithAllInnerExceptions());
            }
        }
    }
}
```

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

```
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.
```

```
13      /// </para>
14      /// <para></para>
15      /// </summary>
16      public class ProgrammerRole
17      {
18          /// <summary>
19          /// <para>
20          /// The git hub api.
21          /// </para>
22          /// <para></para>
23          /// </summary>
24          public readonly GitHubStorage GitHubAPI;
25
26          /// <summary>
27          /// <para>
28          /// The minimum interaction interval.
29          /// </para>
30          /// <para></para>
31          /// </summary>
32          public readonly TimeSpan MinimumInteractionInterval;
33
34          /// <summary>
35          /// <para>
36          /// The triggers.
37          /// </para>
38          /// <para></para>
39          /// </summary>
40          public readonly List<ITrigger<Issue>> Triggers;
41
42          /// <summary>
43          /// <para>
44          /// Initializes a new <see cref="ProgrammerRole"/> 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 ProgrammerRole(List<ITrigger<Issue>> triggers, GitHubStorage gitHubAPI)
57          {
58              GitHubAPI = gitHubAPI;
59              Triggers = triggers;
60              MinimumInteractionInterval = gitHubAPI.MinimumInteractionInterval;
61          }
62
63          /// <summary>
64          /// <para>
65          /// Processes the issues using the specified token.
66          /// </para>
67          /// <para></para>
68          /// </summary>
69          /// <param name="token">
70          /// <para>The token.</para>
71          /// <para></para>
72          /// </param>
73          private void ProcessIssues(CancellationToken token)
74          {
75              while (!token.IsCancellationRequested)
76              {
77                  foreach (var trigger in Triggers)
78                  {
79                      foreach (var issue in GitHubAPI.GetIssues())
80                      {
81                          if (trigger.Condition(issue))
82                          {
83                              trigger.Action(issue);
84                          }
85                      }
86                  }
87                  Thread.Sleep(MinimumInteractionInterval);
88              }
89          }
90
91
```

```
        /// <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 Platform.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 Platform.Bot
{
    /// <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>
```

```
        /// <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

```
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
{
    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";

```

```
22          public void Action(Issue issue)
23          {
24              var issueService = Storage.Client.Issue;
25              var owner = issue.Repository.Owner.Login;
26              var users = GetActivities(GetIgnoredRepositories(Parser.Parse(issue.Body)), owner);
27              StringBuilder sb = new();
28              foreach (var user in users)
29              {
30                  sb.AppendLine($"- **{user.Url.Replace("api.", "").Replace("users/", "")}**");
31                  foreach (var repo in user.Repositories)
32                  {
33                      sb.AppendLine($"  - {repo.Replace("api.", "").Replace("repos/", "")}");
34                  }
35                  // Break line
36                  sb.AppendLine("------------------");
37              }
38              var result = sb.ToString();
39              var comment = issueService.Comment.Create(owner, issue.Repository.Name,
                ↪  issue.Number, result);
40              comment.Wait();
41              Console.WriteLine($"Last commit activity comment is added:
                ↪  {comment.Result.HtmlUrl}");
42              Storage.CloseIssue(issue);
43          }
44
45          public HashSet<string> GetIgnoredRepositories(IList<Link> links)
46          {
47              HashSet<string> ignoredRepos = new() { };
48              foreach (var link in links)
49              {
50                  var values = link.Values;
51                  if (values != null && values.Count == 3 && string.Equals(values.First().Id,
                        ↪  "ignore", StringComparison.OrdinalIgnoreCase) &&
                        ↪  string.Equals(values.Last().Id.Trim('.'), "repository",
                        ↪  StringComparison.OrdinalIgnoreCase))
52                  {
53                      ignoredRepos.Add(values[1].Id);
54                  }
55              }
56              return ignoredRepos;
57          }
58
59          public HashSet<Activity> GetActivities(HashSet<string> ignoredRepositories, string owner)
60          {
61              HashSet<Activity> activeUsers = new();
62              foreach (var repository in Storage.Client.Repository.GetAllForOrg(owner).Result)
63              {
64                  if (ignoredRepositories.Contains(repository.Name))
65                  {
66                      continue;
67                  }
68                  foreach (var commit in Storage.GetCommits(repository.Owner.Login,
                        ↪  repository.Name, DateTime.Today.AddMonths(-3)))
69                  {
70                      if (!activeUsers.Any(x => x.Url == commit.Author.Url))
71                      {
72                          activeUsers.Add(new Activity() { Url = commit.Author.Url, Repositories =
                                ↪  new List<string> { repository.Url } });
73                      }
74                      else
75                      {
76                          if (!activeUsers.Any(x => x.Repositories.Any(y => y == repository.Url)
                                ↪  == true))
77                          {
78                              activeUsers.FirstOrDefault(x => x.Url ==
                                    ↪  commit.Author.Url).Repositories.Add(repository.Url);
79                          }
80                      }
81                  }
82              }
83              return activeUsers;
84          }
85      }
86  }
```

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

```
1   using Interfaces;
2   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));
```

```
79              issueService.Comment.Create(owner, issue.Repository.Name, issue.Number,
                ↪  activeUsersString);
80              Storage.CloseIssue(issue);
81          }
82
83          /// <summary>
84          /// <para>
85          /// Gets the ignored repositories using the specified links.
86          /// </para>
87          /// <para></para>
88          /// </summary>
89          /// <param name="links">
90          /// <para>The links.</para>
91          /// <para></para>
92          /// </param>
93          /// <returns>
94          /// <para>The ignored repos.</para>
95          /// <para></para>
96          /// </returns>
97          public HashSet<string> GetIgnoredRepositories(IList<Link> links)
98          {
99              HashSet<string> ignoredRepos = new() { };
100             foreach (var link in links)
101             {
102                 var values = link.Values;
103                 if (values != null && values.Count == 3 && string.Equals(values.First().Id,
                    ↪  "ignore", StringComparison.OrdinalIgnoreCase) &&
                    ↪  string.Equals(values.Last().Id.Trim('.'), "repository",
                    ↪  StringComparison.OrdinalIgnoreCase))
104                 {
105                     ignoredRepos.Add(values[1].Id);
106                 }
107             }
108             return ignoredRepos;
109         }
110
111         /// <summary>
112         /// <para>
113         /// Gets the active users using the specified ignored repositories.
114         /// </para>
115         /// <para></para>
116         /// </summary>
117         /// <param name="ignoredRepositories">
118         /// <para>The ignored repositories.</para>
119         /// <para></para>
120         /// </param>
121         /// <param name="owner">
122         /// <para>The owner.</para>
123         /// <para></para>
124         /// </param>
125         /// <returns>
126         /// <para>The active users.</para>
127         /// <para></para>
128         /// </returns>
129         public HashSet<string> GetActiveUsers(HashSet<string> ignoredRepositories, string owner)
130         {
131             HashSet<string> activeUsers = new();
132             var date = DateTime.Now.AddMonths(-1);
133             foreach (var repository in Storage.Client.Repository.GetAllForOrg(owner).Result)
134             {
135                 if (ignoredRepositories.Contains(repository.Name))
136                 {
137                     continue;
138                 }
139                 foreach (var commit in Storage.GetCommits(repository.Owner.Login,
                    ↪  repository.Name))
140                 {
141
142                     activeUsers.Add(commit.Author.Login);
143
144                 }
145                 foreach (var pullRequest in Storage.GetPullRequests(repository.Owner.Login,
                    ↪  repository.Name))
146                 {
147                     foreach (var reviewer in pullRequest.RequestedReviewers)
148                     {
149                         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;
        }
    }
}
```

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

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

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

        public void Action(Issue issue)
        {
            var repositories =
            ↪  Storage.Client.Repository.GetAllForOrg(issue.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(issue.Repository.Id, issue.Number,
                ↪  failedRepositoriesComment.ToString());
            }
            else
            {
                Storage.Client.Issue.Comment.Create(issue.Repository.Id, issue.Number, "Success.
                ↪  All repositories default branch protection is updated.");
                Storage.CloseIssue(issue);
            }
        }

        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);
```

```
55                }
56                return result;
57            }
58        }
59    }
```

Index