GitInsight

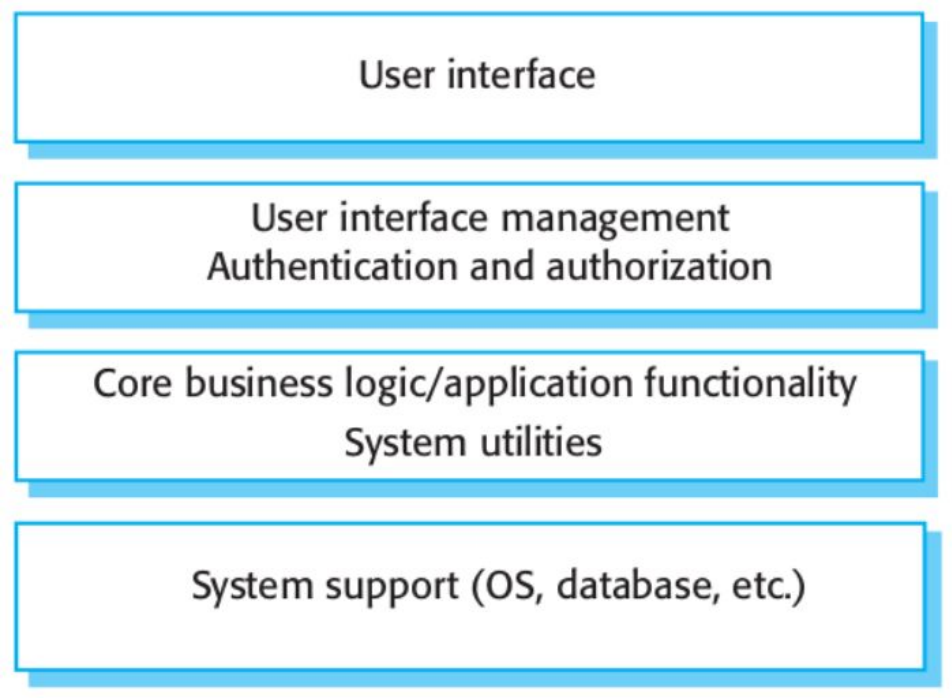
By Ceenja Impact

Architectural Patterns

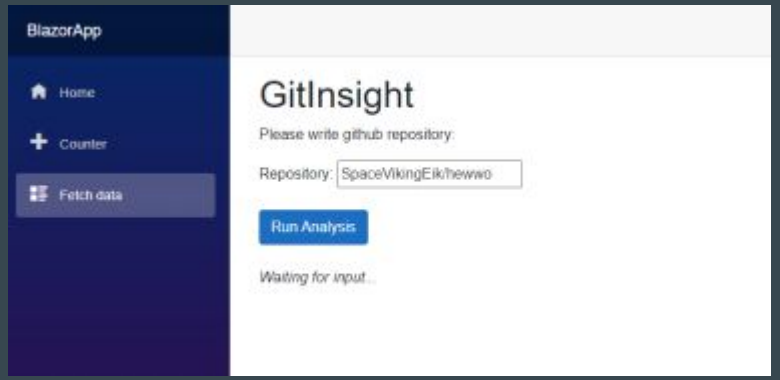
- Client-Server
- Layered
- MVC
- Pipes & Filters
- Repository
- Peer-to-peer

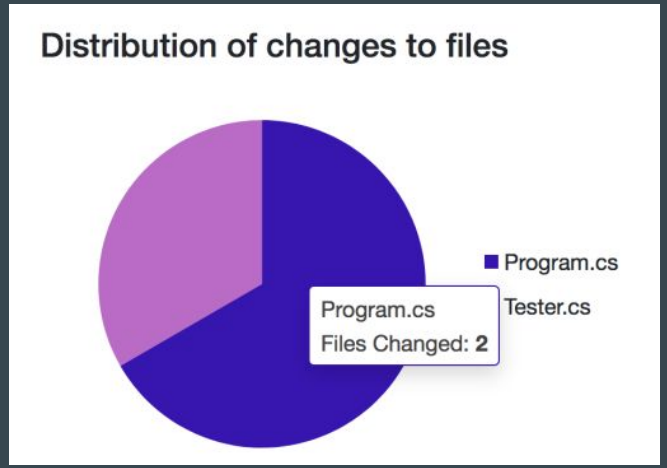
Architectural patterns

Layered Architecture



User Interface





Authentication and Authorization

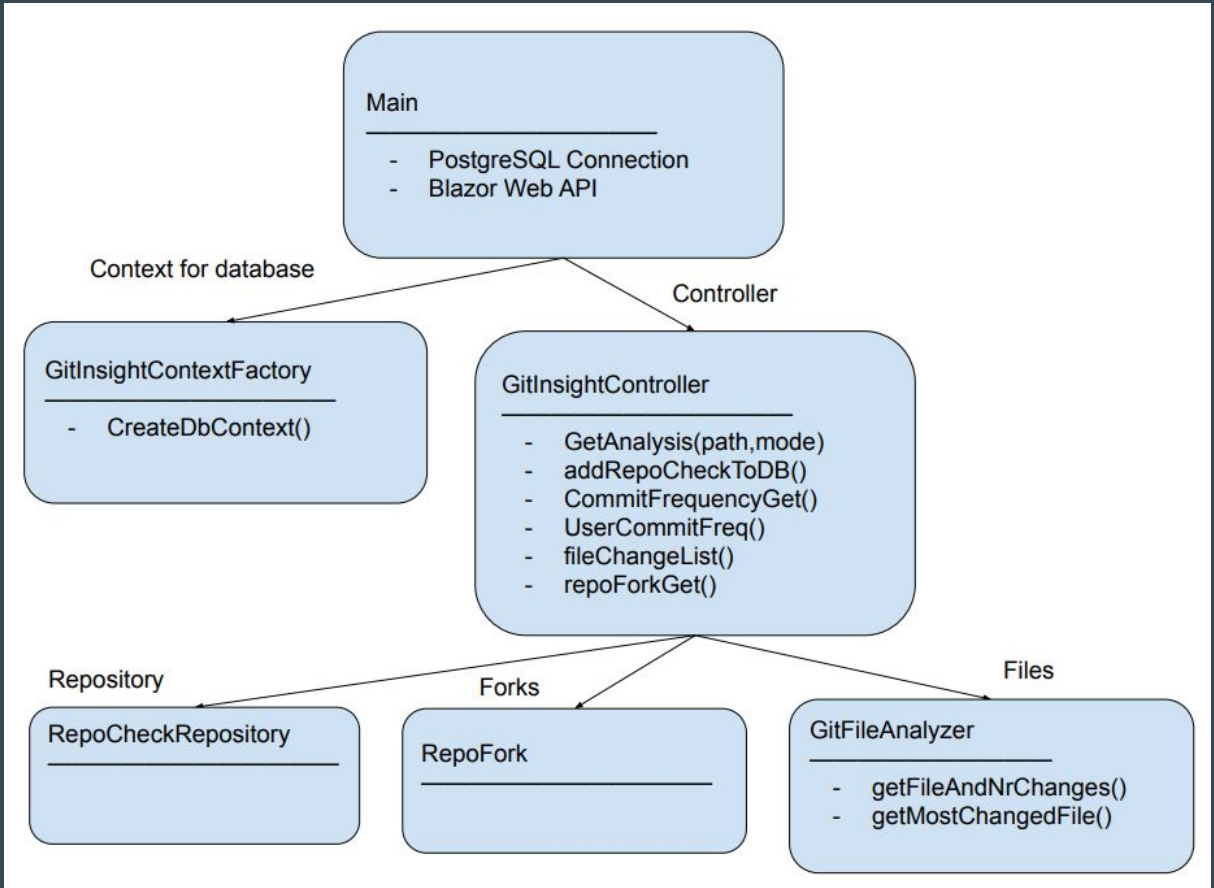
Core logic

```
public List<comFreqObj> getCommitFreq(string t
    _context.Database.OpenConnection();

        var repoCheckItem = _context.RepoChecl
        var items = _context.Contributions.Whe

        var date = items.Select(c => c.date.Da

        var intList = new List<int>();
        foreach(var d in date){
            var comCount = items.Where(k => k.
            .Select(k => k.commitsCount).Sum()
            intList.Add(comCount);
        }

        var tempList = new List<comFreqObj>();
        for (var i = 0; i < intList.Count; i++
            var tem = new comFreqObj(date[i].D
            tempList.Add(tem);
        }

        return tempList;
```

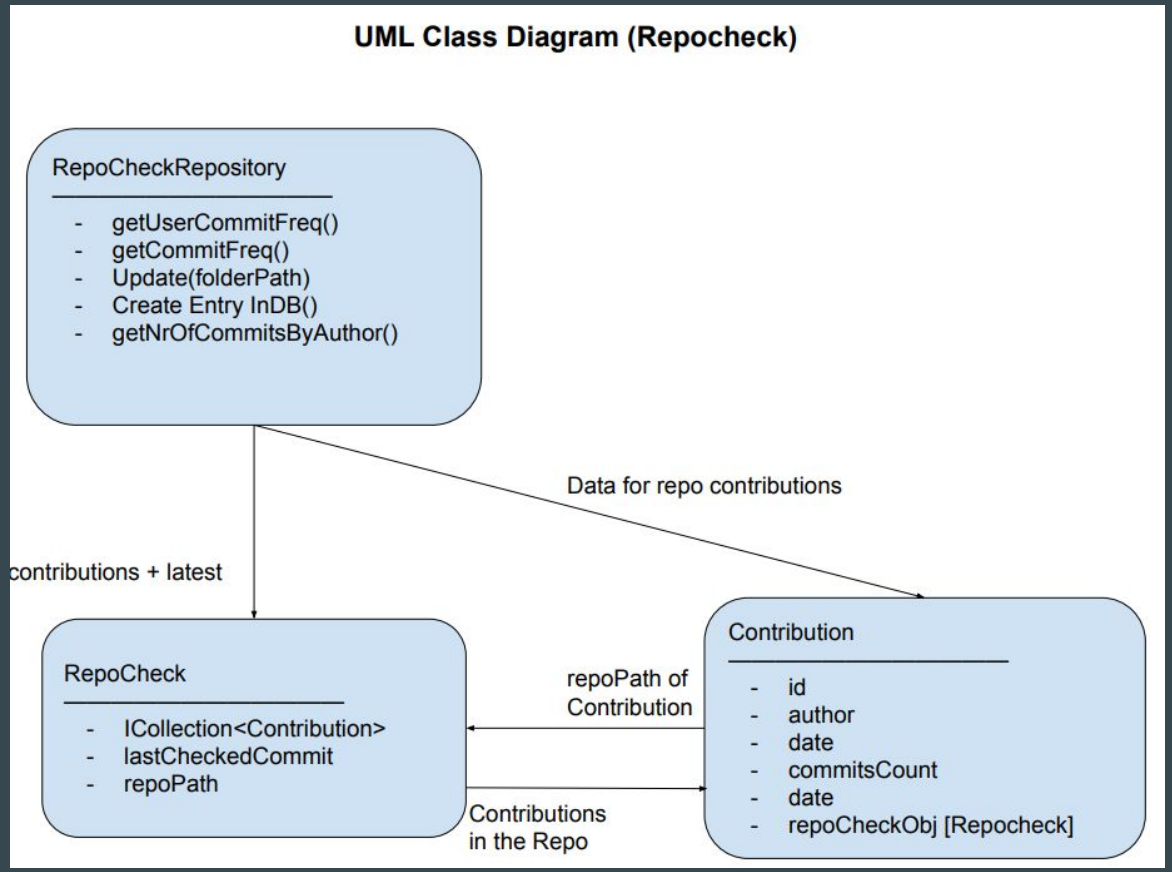
System Support

PostgreSQL

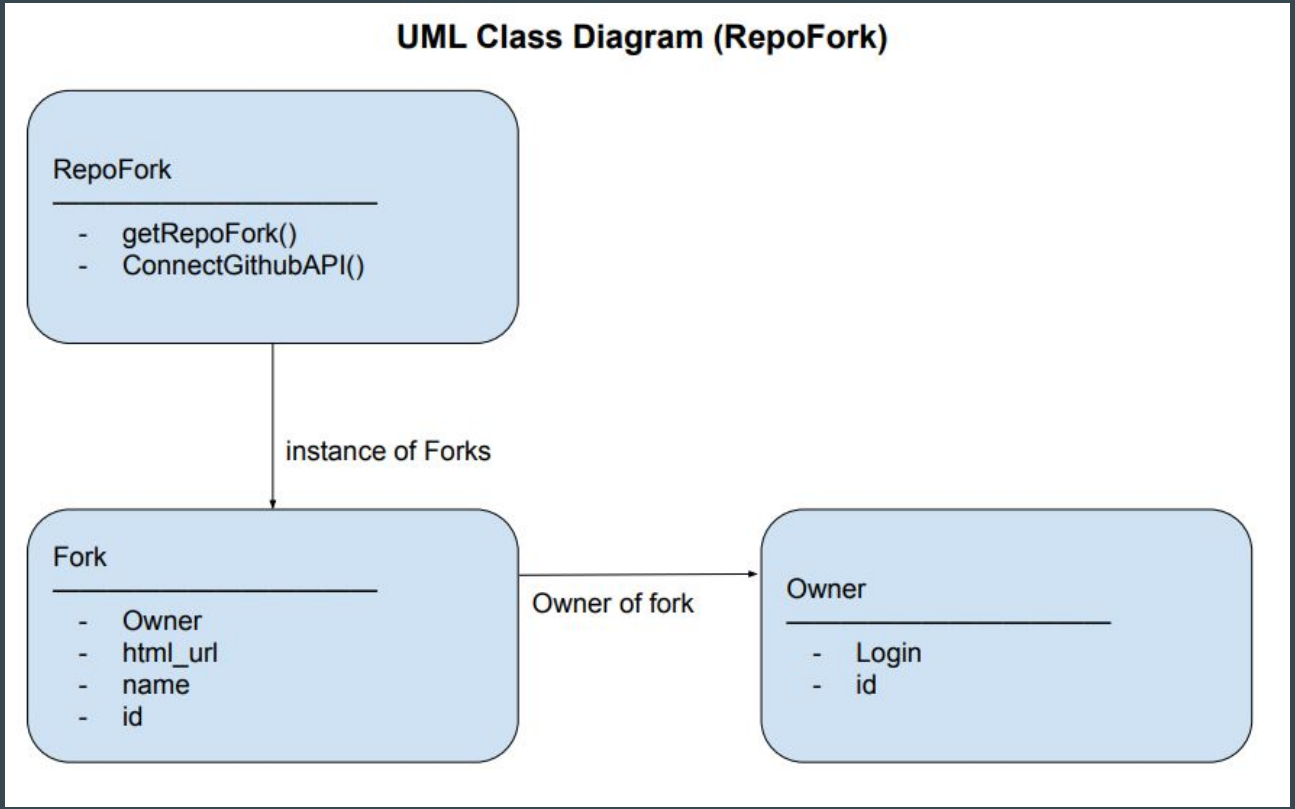
UML Diagrams



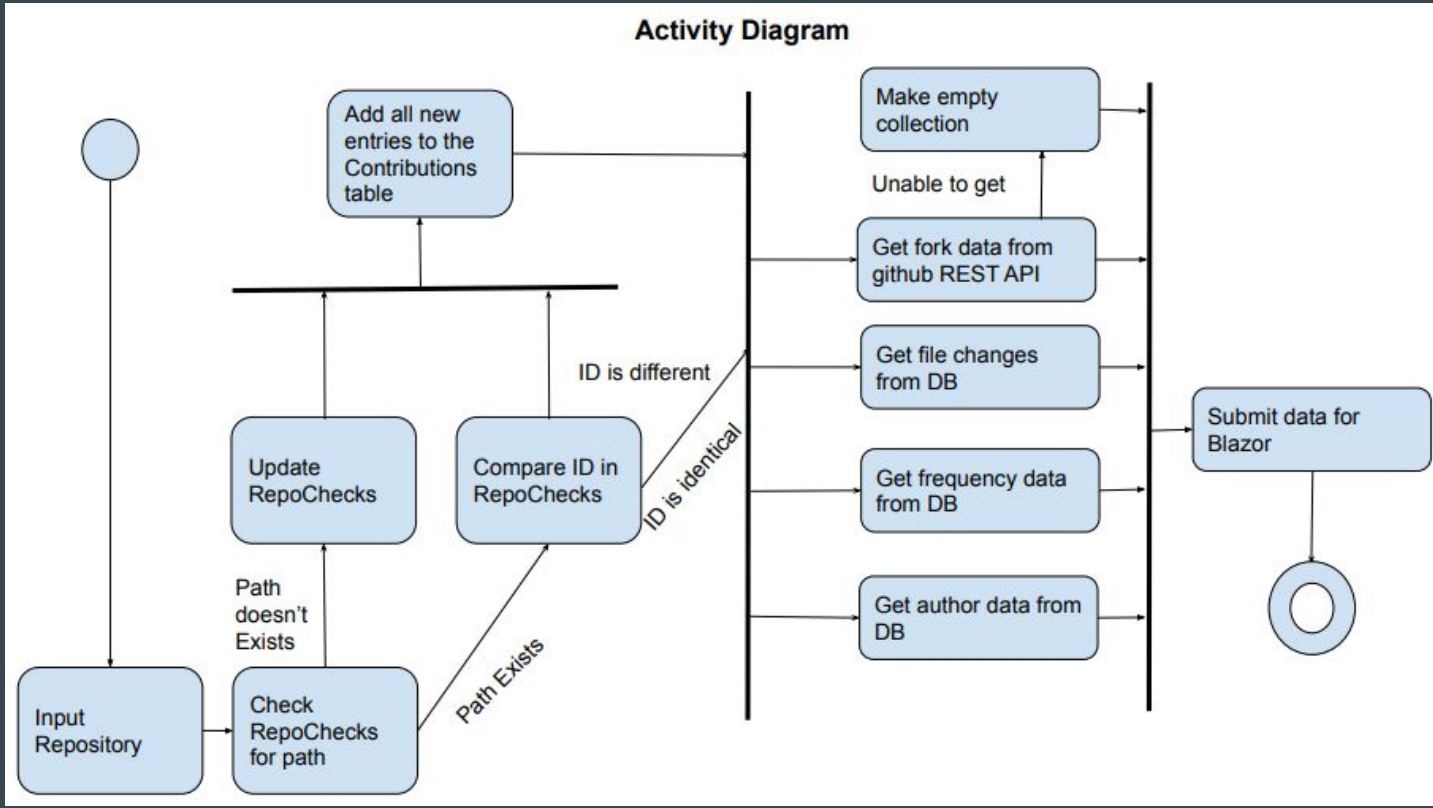
UML Diagrams



UML Diagrams



UML Diagrams



Functional & non-functional requirements

- Iterative project leads to:
  - New Requirements
  - Changing old requirements

Week 1

week 1)
F1: The System should be able to generate a output given any path to a local Git repository.
F2: The User should be able run the program in either "commit frequency" and "commit author" mode
F3: In "commit frequency" mode, the system should show the number of commits made by any author, for each day a commit was made, for the given repository.
F4: In "commit author" mode, the system should show each author who has made a commit, and the number of commits they have made each day, for the given repository.
F5: Dates should be shown in the yyyy-mm-dd format. (subject to change)
F6: Dates should be shown in chronological order. (subject to change)
F7: in "commit author" mode, the authors should be shown in alfabetical order. (subject to change)

Week 2

week 1)
F1: The System should be able to generate a output given any path to a local Git repository.
F2: The User should be able run the program in either "commit frequency" and "commit author" mode
F3: In "commit frequency" mode, the system should show the number of commits made by any author, for each day a commit was made, for the given repository.
F4: In "commit author" mode, the system should show each author who has made a commit, and the number of commits they have made each day, for the given repository.
F5: Dates should be shown in the yyyy-mm-dd format. (subject to change)
F6: Dates should be shown in chronological order. (subject to change)
F7: in "commit author" mode, the authors should be shown in alfabetical order. (subject to change)

week 2 (updated for week 3 changes))

1. When running a quary on the system, it should save all the commit entries read during the quiry, in a database.
   1a) The database will have a single table.
   1a.1) The table will have the repo path as an attribute. 1a.2) The table will have an ID of the newest commit as an attribute.
   1a.3) The table will have a collection of contributions objects. Each contribution object will include "Auther, Date, nr of commits". This will be the objects the system reads to generate the appropriate output.
2. If a quary with a repo that is already in the database, the system should check whether any new commits have been made to the repo since it was added to the database.
   2a) if the repo path doesn't exist in the datebase, it will create an entirely new entry in the table, including the repo path, ID of last commit, and all the contribution objects for the database will be created and saved. 2b) If the repo exists in the database, but haven't had any new commits since last the quiry was run on that repo, then it will just read all contributions objects for the given repo, and generate the appropiate output based on these objects in the database.
   2c) If the repo has had new commits since the last time the quiry was run on that repo, the first thing the system should do, is add all the new entries as contribution objects and save them in the database under the given repo. Afterwards it will generate the appropiate output based on these contribution objects under the specified repo.

Where we got our requirements from

Week One (Week 43)

Build a small C#/.Net Core application that can be run from the command-line. As a parameter, it should receive the path to a Git repository that resides in a local directory, i.e., a directory on your computer.

Given that path to a repository, your application should collect all commits with respective author names and author dates. The data can be collected with the library `libgit2sharp`, which can be installed from NuGet.

Your program should be able to run in two modes, which may be indicated via command-line switches.

F2: The User should be able run the program in either "commit frequency" and "commit author" mode

?

F5: Dates should be shown in the yyyy-mm-dd format. (subject to change)

F6: Dates should be shown in chronological order. (subject to change)

Problems with creating arbitrary requirements

```
F5: Dates should be shown in the dd-mm-yyyy format.
```

```
F5: Dates should be shown in the yyyy-mm-dd format. (subject to change)
```

Problems with creating arbitrary requirements

```
F5: Dates should be shown in the dd-mm-yyyy format.
```

```
F5: Dates should be shown in the yyyy-mm-dd format. (subject to change)
```

Our requirements weren’t shaping out program.
Our program were shaping the requirements.

How we actually decided on our implementation

Additionally, in your `GitInsight` back-end applications implement a new analysis. It should be able to list all forks of a repository on GitHub. To do so, it should call the GitHub REST API and collect a list of all forks from a given repository. That is, when your `GitInsight` REST API receives a `GET` request with a GitHub repository identifier of the form `<github_user>/<repository_name>` or `<github_organization>/<repository_name>`, then besides the two already existing analyses of cloned Git repositories your application contacts the GitHub REST API to collect the number of forks of that repository.

```
//takes the "/repos/:Organization name:/:Repo Name:/forks" as an argument
try{
var gitForks = await client.GetAsync("/repos/" + gitPath + "/forks").ConfigureAwait(false);
gitForks.EnsureSuccessStatusCode();

string responseBody = await gitForks.Content.ReadAsStringAsync().ConfigureAwait(false);
var list = new List<Fork>();

var forkList = JsonConvert.DeserializeObject<List<Fork>>(responseBody);

return forkList!;

}catch(Exception){
    return new List<Fork>();
}
```

Add a front-end web-application that you write with .Net Blazor (WebAssembly) to your already existing applications. That front-end application interacts with your `GitInsight` back-end application via the REST API that you implemented last week.

```
@inherits LayoutComponentBase

<PageTitle>GitInsight</PageTitle>

<div class="page">
    <div class="sidebar">
        <NavMenu />
    </div>

    <main>
        <div class="top-row px-4">
            <a href="https://docs.microsoft.com/aspnet/" target="_blank">About</a>
        </div>

        <article class="content px-4">
            @Body
        </article>
    </main>
</div>
```

Tests

- Unit tests
  - ForkRepo, RepoCheckRepository & GitFileAnalyser
  - Interactions with database - Reading data and returning it in the requested form (fx commit frequency mode)
  - Or: inserting/deleting/manipulating data in db

- Integration testing
  - Blazor server - webpage

Example - Analysis

```
public List<FileAndNrChanges> getFilesAndNrChanges(Repository repo){
    //sort commits by date
    var filter = new CommitFilter {
    SortBy = CommitSortStrategies.Time | CommitSortStrategies.Reverse
    };
    var commits = repo.Commits.QueryBy(filter);

    //Find changes between trees, save changes to list
    var tempList = new List<TreeEntryChanges>();
    for (int i = 0; i < commits.Count(); i++){
        if(i+1 < commits.Count()){        Anne-Marie Roommerdahl, 2 wee
            var temp = repo.Diff.Compare<TreeChanges>(commits.Element
```

```
[Fact]
0 references | Run Test | Debug Test
public void ListOfChangesShouldBeProgram2_Tester1(){

    //Arrange
    var analyser = new GitFileAnalyzer();

    //Act
    var expected = new List<GitFileAnalyzer.FileAndNrChanges>();
    expected.Add(new GitFileAnalyzer.FileAndNrChanges("Program.cs", 2));
    expected.Add(new GitFileAnalyzer.FileAndNrChanges("Tester.cs", 1));

    var actual = analyser.getFilesAndNrChanges(_repo);

    //Assert
    actual.Equals(expected);
}
```

Example - Integrated test

```
[Test]
0 references
public async Task IntegrTest()
{

    await Page.GotoAsync("https://localhost:7111/");

    await Page.GetByRole(AriaRole.Textbox).ClickAsync();

    await Page.GetByRole(AriaRole.Textbox).FillAsync("Divik-kid/BDSA00");

    await Page.GetByRole(AriaRole.Button, new() { NameString = "Run Analysis" }).ClickAsync();
  You, 16 hours ago • fixed a few bugs …
    await Page.Locator(".rz-column-series > path").First.ClickAsync();

    await Page.GetByText("08-09-2022").First.ClickAsync();

    await Page.GetByRole(AriaRole.Heading, new() { NameString = "AuthMode Barchart(s)" }).ClickAsync();

    await Page.GetByText("Chris").ClickAsync();
```

Design patterns- Builder

```
public GitInsightContext CreateDbContext(string[] args)
{
    var configuration = new ConfigurationBuilder().AddUserSecrets<GitInsightClass>().Build();
    var connectionString = configuration.GetConnectionString("GitIn");
    /*$CONNECTION_STRING="Host=localhost;Database=postgres;Username=<username>;Password=<password>");"
    dotnet user-secrets set "ConnectionStrings:GitIn" "$CONNECTION_STRING"*/

    var optionsBuilder = new DbContextOptionsBuilder<GitInsightContext>();
    optionsBuilder.UseNpgsql(connectionString);

    var context = new GitInsightContext(optionsBuilder.Options);
    return context;
}
```

GitInsightContextFactory.cs

```
var builder = WebApplication.CreateBuilder(args);
```

Main.cs

```
if (builder.Environment.IsDevelopment())
{
    app.UseDeveloperExceptionPage();
    app.UseSwagger();
    app.UseSwaggerUI(c => c.SwaggerEndpoint("/swagger/v1/swagger.json", "GetInsight v1"));
}
```

Design principles - SOLID

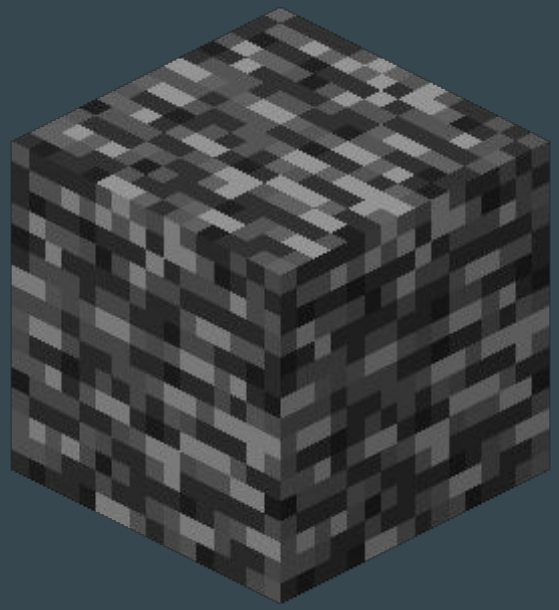
Single Responsibility Principle

Open/Closed Principle

Liskov Substitution Principle

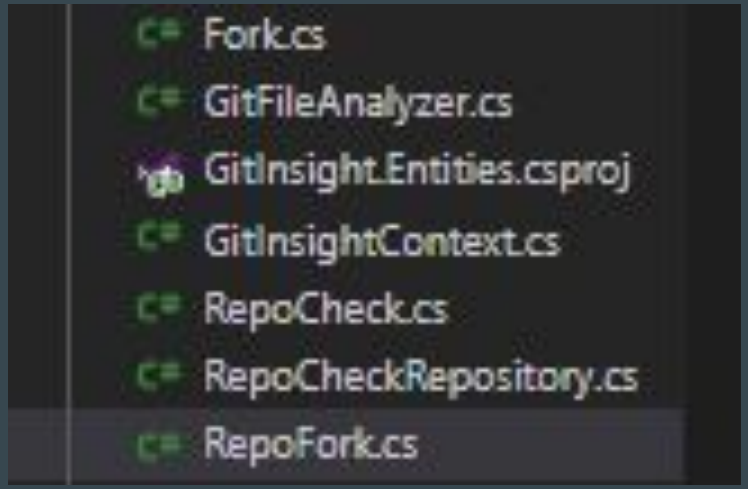
Interface Segregation Principle

Dependency-Inversion Principle



Single Responsibility Principle

```
public class RepoCheckRepository {

    private GitInsightContext _context;

    public RepoCheckRepository(GitInsightContext context)
    {
        _context = context;
    }

    public static Contribution ContributionFromContributionDTO(ContributionDTO contribution)
        => new Contribution
        {
            author = contribution.Author,
            date = contribution.Date,
            commitsCount = contribution.CommitsCount
        };


    public void CreateEntryInDB(string folderPath){

        var repo = new Repository(folderPath);

        var checkedCommit = repo.Commits.ToList().First().Id.ToString();

        var conDTOs = AddContributionsDataToSet(repo);

        var newRepoCheck = new RepoCheck
                {
                    repoPath = folderPath,
                    lastCheckedCommit = checkedCommit,
                    Contributions = conDTOs.Select(c =>
                    ContributionFromContributionDTO(c)).ToHashSet()
                };

        _context.RepoChecks.Add(newRepoCheck);
        _context.SaveChanges();
    }
```



Open/Closed Principle

```
//----------------------Helper methods----------------
    public HashSet<ContributionDTO> AddContributionsDataToSet(Repository repo){
        //add repo data to hashset
        var commitArray = repo.Commits.ToList();
        var contributionsList = new HashSet<ContributionDTO>();

        foreach (var c in commitArray){
            //get number of commits by auhtor on date
            int commitNr = getNrCommitsOnDateByAuthor(c.Author.When.Date, c.Author, repo);

            var newContri = new ContributionDTO(
                Author: c.Author.ToString(),
                Date: c.Author.When.Date, //change to string format dd-mm-yy w. no 00:00:00!
                CommitsCount: commitNr);

            contributionsList.Add(newContri);
        }

        return contributionsList;
    }

    public HashSet<ContributionDTO> AddContributionsDataToSetButRemoveEverythingAlreadyThere(Repository repo, String folderPath){
        //add repo data to hashset
        var commitArray = repo.Commits.ToList();
        var contributionsList = new HashSet<ContributionDTO>();
        //just for testing remove later
        Console.WriteLine(commitArray.Count);
        Console.WriteLine(repo.Info.Path);

        var repoCheckItem = _context.RepoChecks.Find(folderPath); //check om commit newest, fix
        var contributions = _context.Contributions.Where(c => c.repoCheckObj!.Equals(repoCheckItem));
        var latestCommitDate = contributions.Select(c => c.date).Max();
        var lastCommitList = contributions.Where(c => c.date.Equals(latestCommitDate)).ToList();

        foreach(var d in lastCommitList){
            var toDel = _context.Contributions.Find(d.Id);
            _context.Contributions.Remove(toDel!);
        }

        foreach (var c in commitArray){
            Console.WriteLine(c.Author.When.Date);
            //get number of commits by auhtor on date
            if(c.Author.When.Date >= latestCommitDate){
                int commitNr = getNrCommitsOnDateByAuthor(c.Author.When.Date, c.Author, repo);

                var newContri = new ContributionDTO(
                    Author: c.Author.ToString(),
                    Date: c.Author.When.Date,
                    CommitsCount: commitNr);

                contributionsList.Add(newContri);
            }


        }

        return contributionsList;
    }

    private int getNrCommitsOnDateByAuthor(DateTime date, Signature author, Repository repo){
        var commitsCount = repo.Commits
        .Select(e => new { e.Author, e.Author.When.Date })
        .Where(e => e.Author.ToString() == author.ToString()
        && e.Author.When.Date == date).Count();

        return commitsCount;
    }

//-------------------------------------------------
```

```
public HashSet<ContributionDTO> ContributionToContributionDTOHS(RepoCheck repoCheck){
    var contributions = repoCheck.Contributions
                .Select(cont => new ContributionDTO(
                    cont.author!,
                    cont.date, cont.commitsCount
                )).ToHashSet();
    return contributions;
}

//only use this in testing for now
public RepoCheckDTO Read(string folderPath){
    var repoCheck = _context.RepoChecks.Find(folderPath);
    //var DTO = new RepoCheckDTO();
    return repoCheck != null ? new RepoCheckDTO(
                                repoCheck.repoPath!,
                                repoCheck.lastCheckedCommit!,
                                Contributions: ContributionToContributionDTOHS(repoCheck)) : null!;
}

public List<comFreqObj> getCommitFreq(string folderPath){
    _context.Database.OpenConnection();

        var repoCheckItem = _context.RepoChecks.Find(folderPath); //check om commit newest, fix
        var items = _context.Contributions.Where(c => c.repoCheckObj!.Equals(repoCheckItem));

        var date = items.Select(c => c.date.Date).Distinct().ToList();

        var intList = new List<int>();
        foreach(var d in date){
            var comCount = items.Where(k => k.date.Date.Equals(d))
            .Select(k => k.commitsCount).Sum();
            intList.Add(comCount);
        }

        var tempList = new List<comFreqObj>();
        for (var i = 0; i < intList.Count; i++){
            var tem = new comFreqObj(date[i].Date.ToShortDateString(), intList[i]);
            tempList.Add(tem);
        }

        return tempList;


}
```

Liskov Substitution Principle

Throw new NotImplementedException()

```
18 references
public partial class GitInsightContext : DbContext
{
    2 references
```

```
5 references
public class GitInsightController : ControllerBase
{
    2 references
```

Interface Segregation Principle

Throw new NotImplementedException()

We have simply not used interfaces

Dependency-Inversion Principle

Throw New NotFollowedException()

```
5 references
public class GitInsightController : ControllerBase
{
    2 references
    private GitInsightContext _context;
    6 references
    private RepoCheckRepository _repository;
    2 references
    private GitFileAnalyzer _analyzer;

    2 references
    public GitInsightController(GitInsightContext context)
    {
        _context = context;
        _repository =  new RepoCheckRepository(context);
        _analyzer = new GitFileAnalyzer();
        _context.Database.OpenConnection();
    }
```

```
    //--Helper method to GetAnalysis()-----
    1 reference
    private void addRepoCheckToDB(string repoPath){
        //create repocheck and add in _repository
        _repository.CreateEntryInDB(repoPath);
    }
    //---------------------------------

    1 reference
    private List<RepoCheckRepository.comFreqObj> CommitFrequencyGet(string folderPath){
        return _repository.getCommitFreq(folderPath);
    }

    1 reference
    private List<RepoCheckRepository.userComFreqObj> userCommitFreq(string folderPath){
        return _repository.getUserCommitFreq(folderPath);
    }

    1 reference
    private List<GitFileAnalyzer.FileAndNrChanges> fileChangeList(string folderPath){
        var repo = new Repository(folderPath);
        return _analyzer.getFilesAndNrChanges(repo);
    }

    1 reference
    private List<RepoFork.RepoForkObj> repoForkGet(string folderPath){
            return RepoFork.getRepoForks(folderPath, null).GetAwaiter().GetResult().ToList();
    }

}
```