



**Bilkent University**  
**Department of Computer Engineering**

**Senior Design Project**  
*T2310*  
*ReLink*

## **Analysis and Requirement Report**

*Ayberk Yaşı, 21801847, ayberk.yasa@ug.bilkent.edu.tr*  
*Cemhan Kaan Özaltan, 21902695, kaan.ozaltan@ug.bilkent.edu.tr*  
*Çağatay Şafak, 21902730, cagatay.safak@ug.bilkent.edu.tr*  
*Fatih Kaplama, 21802755, fatih.kaplama@ug.bilkent.edu.tr*  
*Gökem Ayten, 21802399, gorkem.ayten@ug.bilkent.edu.tr*

*Eray Tüzün*

*Erhan Dolak, Tağmaç Topal*

**13th of November 2022**

This report is submitted to the Department of Computer Engineering of Bilkent University in partial fulfillment of the requirements of the Senior Design Project course CS491/2.

# Table of Contents

1. Introduction	4
2. Proposed System	4
2.1. Overview	4
2.1.1. User Types	5
2.1.2. Sign Up & Sign In	5
2.1.3. Connect Project to ReLink	6
2.1.4. Recovering Missing Links in the Past	6
2.1.5. Historical Graph Visualization	6
2.1.6. Visualization of Analysis of Missed Links	6
2.1.7. Link Recommendations	6
2.1.8. Warning Message	7
2.1.9. Categorizing Source Code Files	7
2.2 Functional Requirements	7
2.3. Nonfunctional Requirements	9
2.3.1. Usability	9
2.3.2. Reliability	9
2.3.3. Performance	10
2.3.4. Supportability	10
2.3.5. Scalability	10
2.4. Pseudo Requirements	10
2.4.1. Implementation	10
2.5. System Models	10
2.5.1. Scenarios	10
2.5.2. Use Case Model	18
2.5.3. Object and Class Model	19
2.5.4. Dynamic Models	21
2.5.4.1. Create Project Activity Diagram	21
2.5.4.2. Create Instant Link Activity Diagram	22
2.5.4.3. Recover Missing Links Activity Diagram	23
2.5.4.4. Link Item State Diagram	24
2.5.4.5. Commit State Diagram	25
2.5.5. User Interface - Navigational Paths and Screen Mock-ups	26
2.5.5.1. Navigational Path	26
2.5.5.2. Mock-ups	26
2.5.5.2.1. Login and Register Page	26
2.5.5.2.2. New Project Page	27
2.5.5.2.3. Projects Page	28
2.5.5.2.4. Overview Page	29
2.5.5.2.5. Analysis Page	29
2.5.5.2.6. Missing Links Page	31
2.5.5.2.7. Historical View Page	32
2.5.5.2.8. File Categorization Page	33

2.5.5.2.9. Create Instant Link Page	34
2.5.5.2.10. Warning Page	34
3. Other Analysis Elements	35
3.1. Consideration of Various Factors in Engineering Design	35
3.2. Risks and Alternatives	35
3.3. Project Plan	37
3.4. Ensuring Proper Teamwork	37
3.5. Ethics and Professional Responsibilities	38
3.6. Planning for New Knowledge and Learning Strategies	38
4. Glossary	38
5. References	39

# 1. Introduction

In our age of technology, bugs, improvements, and features of a software project are stored and tracked in project management tools that provide development teams with an issue tracker. The rise of agile results in large amounts of data about issues in these project management tools which require much more effort to inspect manually [1]. Moreover, commits and pull requests are fundamental artifacts used in the software development lifecycle (SDLC) [1], [2]. Whereas issues are stored in bug tracking and project management tools such as Jira, Bugzilla, and GitHub Issues, pull requests and commits are stored in servers utilizing a version control system (VCS) such as GitHub, GitLab, and Bitbucket. Such contemporary VCS options offer built-in features for manually linking issues with commits and pull requests through a predefined commit message syntax where the issue ID is explicitly given, allowing for a more structured and traceable project. However, despite the importance of traceability in software development, most commits and pull requests are not explicitly linked to issues by developers due to the lack of the fixed rules [3]. The information contained in the missing links is therefore lost, reducing efficiency in processes such as bug localization, bug prediction, and feature improvements [3]. Since it is highly possible for developers to forget establishing such links or fail tagging a commit due to a typo and the manual cost of recovering these links is very high, we propose an automation tool for tackling this issue.

All the user types and the features that will be found in the project are explained in detail in section 2.1 to make the outline of the project more comprehensible. Section 2.2 describes what kind of actions actors can do and lists them. Section 2.3 describes some constraints affecting the system's and the user's behavior. Pseudo requirements are discussed in Section 2.4. Section 2.5 lists the scenarios, use case diagram, class diagram, activity diagrams, and some user interface mock-ups with a navigational path. Section 3 includes other analysis elements. Finally, the context-specific terms used through the analysis report and references are provided in Section 4 and 5, respectively.

## 2. Proposed System

### 2.1. Overview

ReLink is a web application which serves as a tool for automating the issue-commit and issue-pull request linking process during software development. To be specific, ReLink detects missing links retrospectively and creates a link between issue-commit pairs or issue-pull request pairs. Moreover, ReLink determines whether a link is constructed between the commit and an issue at the commit stage. If the developers forget to tag the commit with an issue ID, this tool warns the developers and suggests possible issues that may be related to the corresponding commit. This

process is also available for opening a pull request. All these are done through certain machine learning (ML) algorithms. ReLink also provides the visualization of the analysis of missed links among commits, pull requests, and issues by certain filters such as by developer, by time, and by reason. Finally, ReLink offers a graphical interface for visualizing data on the historical evolution and progress of a specific project's issues. To be specific, there is a historical graph visualization for links among commits, pull requests, and issues.

### 2.1.1. User Types

The system has three kinds of users:

- Admin
- Developer
- Project Manager

**Admin** should be able to manage every user and project on the application, just in case if there occurs an issue. Moreover, only the admin should be able to assign a project manager for each project.

**Developer** should be able to only view the pages related to a project and not be able to make any modifications. However, Developer should be able to receive warning messages, get recommendations, and create links at the commit and pull request creation stage.

**Project Manager** should be able to do everything that a developer can do. Additionally, Project Manager should be able to manage the projects, analyzes, and historical graphs. They, also, should have extra permissions compared to a developer, like adding developers to a project, removing developers from a project and authorizing developers to manage projects like Project Manager . Finally, they should be able to categorize the source code file to help train model and fix missed links in the past.

### 2.1.2. Sign Up & Sign In

The project will have sign up and sign in functionalities. The users will be able to do these operations with a unique email and password combination associated with their account. These accounts can have different levels of authorization for different kinds of projects. For example, there are different project managers and developers for different projects even in the same company. Project Managers can sign in to the web application with the email and password assigned by ReLink to them. They have to change their password when they sign in to their account for the first time. Developers need to sign up to create a new account. ReLink doesn't assign an account for them.

### **2.1.3. Connect Project to ReLink**

Project Manager should provide a link to a project that he/she works on. ReLink needs the GitHub or Azure DevOps repository and Github Issues, Azure DevOps, or JIRA links of the project to read, analyze and change the commit, pull requests and issue links. If the project is publicly available, only url information is sufficient. Extra credentials may be requested when the repository is private.

### **2.1.4. Recovering Missing Links in the Past**

The users will be able to access a dedicated page where the previous commits and PRs made for a specific project, which are not explicitly linked to an issue by the developer who created them can be observed and suitable links may be created for them. This core functionality of ReLink will be given to the user as a recommender system, where for each unlinked commit or PR, the user will be given a list of options determined with a ML model and will be allowed to choose among the options. The unlinked commits and PRs will be shown as a list, each with their own category, and the user will be able to view further details and make the said decision by clicking on a listed item.

### **2.1.5. Historical Graph Visualization**

Project Managers update the graph visualization between commits, pull requests, and issues by taking a snapshot of the project at a specific time. Developers will be able to view this graph representation of the connections between commits, pull requests, and issues, where the connections were created by an ML algorithm. In addition, this page will include a data table that shows who has linked which commits, pull requests, or issues. Therefore, Project Managers will be able to track the history of the linkage process.

### **2.1.6. Visualization of Analysis of Missed Links**

This feature of the application is to create plots, graphs, and charts of analysis of missed links. Thanks to this feature, the user can inspect the results of analysis in a more concrete way. While showing results to the user, the user can filter the plots, graphs, and charts by developer, time, and reason. Moreover, the user can follow the progression of the status of the missed links in his/her project since repository mining will be performed regularly and missed links will be detected each time.

### **2.1.7. Link Recommendations**

That feature of the application is to be used when Developers try to open a pull request. At the “Create pull request” screen, ReLink provides a brief list of issues which may be possibly related to that pull request. The same feature is available for

Developers who commit their code. ReLink offers a list of issues for the user at the “commit and push changes” screen.

That feature works by counting the number and quality of pull requests and issues, i.e., what will be listed. Then it makes the underlying assumption by analyzing the content of the code in the commit, name of the commit, assignees of the issue, date of the commit, and the issue created to determine a rough estimate of which issue or pull request seems more related to the work. If the user selects an assumption from the list, then the description of the commit should be changed by following the convention determined by the company, or organization. For example, it adds “ABC-001” to the commit description when the user selects the issue ABC-001 on the list. This also applies to the pull request case.

### **2.1.8. Warning Message**

Naming the commits, pull requests, and issues is essential to software project development. When a typo or another problem is related, developers should be warned by a simple pop-up. That is important since one of the main goals of ReLink is to prioritize a convention throughout the development process of a software project. Companies, or organizations, should be able to determine their own convention for commit, issue, and PR names and descriptions, and if the user does not follow those rules, then ReLink will warn them. Those warnings also improve the user experience even when the company provides no such convention.

### **2.1.9. Categorizing Source Code Files**

Categorizing the source code files in the repository of the project is essential for ReLink to give more accurate suggestions. Therefore, Project Manager should categorize the source code file to help train the ML model. This increases the possibility of a prediction with higher accuracy when the commit or PR is not manually linked.

## **2.2 Functional Requirements**

- Admin should be able to assign a project manager for each project.
- Admin should be able to manage(create, read, update, delete) users in the system.
- Admin should be able to manage(create, read, update, delete) projects of the users in the system.
- Project Managers should be able to log in, and log out the system.
- Developers should be able to sign up, log in, and log out the system.
- Developers and Project Managers should be able to change the password.

- Developers and Project Managers should be able to receive a warning message when there is a typo or technical thing that prevents linking between a PR and an issue.
- Developer and Project Manager should be able to receive a warning message when there is a typo or technical thing that prevents linking between commit and issue.
- Developers and Project Managers should be given recommendations for linking PRs to issues if they do not provide an explicit link in PR titles or descriptions through a specific convention, e.g., SA-123.
- Developers and Project Managers should be given recommendations for linking commits to issues if they do not provide an explicit link in commit titles or descriptions through a specific convention, e.g., SA-123.
- Developer and Project Manager should be able to create a link between a PR and issues at the pull request creation stage.
- Developer and Project Manager should be able to create a link between a commit and issues at the commit stage.
- Developer and Project Manager should be able to see the visualization of the analysis of missed links between commits and issues.
  - By developer
  - By time
  - By reason, e.g., a typo, missing, or technical things
- Developers and Project Managers should be able to see the visualization of the analysis of missed links between PRs and issues.
  - By developer
  - By time
  - By reason, e.g., a typo, missing, or technical things
- Developers and Project Managers should be able to see a historical graph visualization of links among commits, pull requests, and issues, where the links are constructed by an ML algorithm.
- Project Manager should be able to categorize the source code file for the sake of training to allow prediction with higher accuracy when the commit or PR is not manually linked.
- Project Manager should be given missed links between PRs and issues in the past through possible match suggestions.
  - Using code similarity and other metadata
  - Using commit messages and NLP
  - Using past correctly matched links
- Project Manager should be able to create links between past PRs and issues by selecting the appropriate match suggestion.
- Project Manager should be able to create and delete a project.
- Project Manager should be able to create and delete an analysis of missed links.



- Project Manager should be able to update the historical graph visualization of links among commits, pull requests, and issues.
- The Project Manager should be able to give a developer access to the project.
- The Project Manager should be able to deprive a developer of access to the project.
- Developer and Project Manager should be suggested a list of possible issues when clicking a commit button or a pull request creation button with an average of 95% recall.
- The accuracy of the automated traceability linking algorithm will be at least 60% accurate.
- The system should be able to support GitHub and AzureDevOps as a version control system and Jira, GitHub Issues, and Azure DevOps as a bug tracking system.

## **2.3. Nonfunctional Requirements**

### **2.3.1. Usability**

- There should exist a navigational sidebar visible from all pages for allowing the user to easily navigate the app.
- The titles on the navigation bar, the titles on each page, and the labels on all buttons should be meaningful and self-explanatory so that users who do not read the user manual are able to understand how the website is used from the titles on the navigation bar, the titles on the screen and the labels on the buttons.
- All screens including pop-ups should be reached by being clicked at most twice.
- In order to be a user-friendly website, it is imperative to be consistent in the website layout and design by following a design pattern in which all screens accessed from the sidebar use the same main template.
- Except for the pop-ups, none of the screens on the website must be connected to each other, so users don't have to go backward on the website.
- Web Content Accessibility Guidelines [4] must be followed, which makes content in the website more accessible to users with disabilities.

### **2.3.2. Reliability**

- Users must be able to access a historical graph visualization for links among commits, pull requests, and issues of their projects 90% of the time without failure.
- The data fetched from repositories must be used without changing.
- Users who close the browser without properly logging out of their account will be automatically logged out.

### 2.3.3. Performance

- The load time of each page must be less than 2 seconds.
- The average response time of each button must be about 2 seconds.
- The website must keep the above-mentioned times the same, up to 100 simultaneous users.

### 2.3.4. Supportability

- User feedback will be evaluated continuously and if there is any bug reported by users, it will be assigned to a developer within 24 hours.

### 2.3.5. Scalability

- When the daily traffic of this website, which has 1000 visitors per day, exceeds this number, the max bandwidth limit of this website that is allocated to the hosting plan should not be exceeded.

## 2.4. Pseudo Requirements

### 2.4.1. Implementation

- Django [5] must be used at the backend side.
- React.js [6] must be used at the front-end side.
- For graph construction and content delivery, the MinIO [7] object storage must be used.
- MongoDB [8] must be used for data storage.
- PyTorch [9] must be used for training existing state-of-the-art models such as BERT [10], in addition to building any custom neural networks.

## 2.5. System Models

### 2.5.1. Scenarios

---

**Use Case Name:** AssignProjectManager

**Participating Actors:** Initiated by Admin

**Flow of Events:**

1. Admin selects the project whose project manager will be assigned.
2. Admin selects a project manager to be assigned to the project.
3. System receives the new role of the selected person and saves it.

**Entry Conditions:** Admin enters the dashboard.

**Exit Conditions:** Admin changes the role of a person.

**Quality Requirements:** A project must have at least one project manager.

---

**Use Case Name:** ManageUsers

**Participating Actors:** Initiated by Admin

**Flow of Events:**

1. Admin selects a user to change its status in the system.
2. Admin updates the status of the selected user.
3. System receives the new status of the selected person and saves it.

**Entry Conditions:** Admin enters the dashboard.

**Exit Conditions:** Admin changes the status of the user in the system.

**Quality Requirements:** Attributes of a user cannot be empty.

---

**Use Case Name:** ManageProjects

**Participating Actors:** Initiated by Admin

**Flow of Events:**

1. Admin selects a project to change its status in the system.
2. Admin updates the status of the selected project.
3. System receives the new status of the selected project and saves it.

**Entry Conditions:** Admin enters the dashboard.

**Exit Conditions:** Admin changes the status of the project in the system.

**Quality Requirements:** Attributes of a project cannot be empty.

---

**Use Case Name:** ReceiveWarning

**Participating Actors:** Initiated by Developer

**Flow of Events:**

1. Developer finishes writing the code.
2. Developer tries to send the code written.
3. System checks the fields Developer filled.

**Entry Conditions:** Developer finishes writing the code.

**Exit Conditions:** Developer receives a warning.

**Quality Requirements:** Warning message should be displayed.

---

**Use Case Name:** ReceiveWarningForPR

**Participating Actors:** Communicates with ReceiveWarning

**Flow of Events:**

1. Developer creates a pull request without an issue number in the title.
2. The system checks the existence of a link to the related issue.
3. The system sends a warning to the Developer.

**Entry Conditions:** User creates a pull request.

**Exit Conditions:** User receives a warning.

**Quality Requirements:** Warning message should be displayed.

---

**Use Case Name:** ReceiveWarningForCommit

**Participating Actors:** Communicates with ReceiveWarning

**Flow of Events:**

1. Developer creates a commit without an issue number in the title.
2. The system checks the existence of a link to the related issue.
3. The system sends a warning to the Developer.

**Entry Conditions:** Developer creates a commit.

**Exit Conditions:** Developer receives a warning.

**Quality Requirements:** Warning message should be displayed.

---

**Use Case Name:** CreateInstantLink

**Participating Actors:** Initiated by Developer

**Flow of Events:**

1. Developer inspects the link recommendations.
2. Developer selects the true link suggested by the system.
3. Developer lets the system add a link.

**Entry Conditions:** Developer opens a version control tool.

**Exit Conditions:** The system adds a specified link.

**Quality Requirements:** The artifact should contain only one link.

---

**Use Case Name:** CreateInstantPRLink

**Participating Actors:** Communicates with CreateInstantLink

**Flow of Events:**

1. Developer inspects the link recommendations when opening a pull request.
2. Developer selects the issue related to the pull request.
3. The system adds the issue number to the pull request's title.

**Entry Conditions:** Developer opens a pull request.

**Exit Conditions:** The system adds a specified link.

**Quality Requirements:** The pull request title should contain only one link.

---

**Use Case Name:** CreateInstantCommitLink

**Participating Actors:** Communicates with CreateInstantLink

**Flow of Events:**

1. Developer inspects the link recommendations when creating a commit.
2. Developer selects the issue related to the commit.
3. The system adds the issue number to the commit's title.

**Entry Conditions:** Developer creates a commit.

**Exit Conditions:** The system adds a specified link.

**Quality Requirements:** The commit title should contain only one link.

---

**Use Case Name:** InspectAnalysis

**Participating Actors:** Initiated by Developer

**Flow of Events:**

1. Developer selects an analysis from the list.
2. Developer opens the analysis.
3. Developer inspects the analysis.

**Entry Conditions:** Developer opens an analysis.

**Exit Conditions:** Developer closes an analysis.

**Quality Requirements:** Developers can inspect one analysis at a time.

---

**Use Case Name:** InspectHistoricalGraph

**Participating Actors:** Initiated by Developer

**Flow of Events:**

1. Developer selects a historical graph from the list.
2. Developer opens the historical graph.
3. Developer inspects the historical graph.

**Entry Conditions:** Developer opens a historical graph.

**Exit Conditions:** Developer closes a historical graph.

**Quality Requirements:** Developers can inspect one graph at a time.

---

**Use Case Name:** GetRecommendation

**Participating Actors:** Initiated by Developer

**Flow of Events:**

1. Developer triggers the system through either a pull request or a commit.
2. The system runs an algorithm to find proper recommendations.
3. The system displays recommendations found.

**Entry Conditions:** Developer opens a version control tool.

**Exit Conditions:** The system provides recommendations for links.

**Quality Requirements:** The system should provide at least one recommendation.

---

**Use Case Name:** GetPRRecommendation

**Participating Actors:** Communicates with GetRecommendation

**Flow of Events:**

1. Developer opens a pull request.
2. The system detects possible issues related to the pull request.
3. The system displays recommendations found.

**Entry Conditions:** Developer opens a pull request.

**Exit Conditions:** The system provides recommendations for links.

**Quality Requirements:** The system should provide at least one recommendation.

---

**Use Case Name:** GetCommitRecommendation

**Participating Actors:** Communicates with GetRecommendation

**Flow of Events:**

1. Developer creates a commit.
2. The system detects possible issues related to the commit.
3. The system displays recommendations found.

**Entry Conditions:** Developer creates a commit.

**Exit Conditions:** The system provides recommendations for links.

**Quality Requirements:** The system should provide at least one recommendation.

---

**Use Case Name:** CategorizeFile

**Participating Actors:** Initiated by Project Manager

**Flow of Events:**

1. Project Manager selects a category.
2. Project Manager selects the corresponding source code files for this category.
3. Project Manager gives keywords possibly related to the source code files.

**Entry Conditions:** Project Manager starts the selection of the category.

**Exit Conditions:** The system saves the categorization of the file.

**Quality Requirements:** Each source code file should have at least one category and keyword.

---

**Use Case Name:** RecoverMissingLinks

**Participating Actors:** Initiated by Project Manager

**Flow of Events:**

1. Project Manager selects a pull request.
2. Project Manager inspects the recommendations.
3. Project Manager selects the issue(s) really related to the pull request.
4. Project Manager triggers the recovering process.

**Entry Conditions:** Project Manager inspects the pull request without any link.

**Exit Conditions:** The system adds the id of the selected issue to the title.

**Quality Requirements:** The selected pull request should have at least one issue link.

---

**Use Case Name:** CreateProject

**Participating Actors:** Initiated by Project Manager

**Flow of Events:**

1. Project Manager enters the title and description of the project.
2. Project Manager selects the data source types.
3. Project Manager adds one or more repository link for each data source type.

4. Project Manager adds an authentication token if the repository is private.

**Entry Conditions:** Project Manager starts creating a project.

**Exit Conditions:** The system saves the project and the corresponding repositories.

**Quality Requirements:** The project should have at least one version control system link and one bug tracking system link.

---

**Use Case Name:** GetPossibleMissingLinks

**Participating Actors:** Initiated by Project Manager

**Flow of Events:**

1. The system detects missing links in the past.
2. The system runs a ML algorithm to recommend issues.
3. The system finds the issues possibly related to each pull request.

**Entry Conditions:** Project Manager enters to see missing links in the past to recover.

**Exit Conditions:** The system shows the possible matches.

**Quality Requirements:** Each pull request has at least one possible issue match.

---

**Use Case Name:** DeleteProject

**Participating Actors:** Initiated by Project Manager

**Flow of Events:**

1. Project Manager selects a project to remove.
2. Project Manager deletes the project.
3. The system receives the removal of that project.

**Entry Conditions:** Project Manager enters the project list.

**Exit Conditions:** The system saves the project list without that project.

**Quality Requirements:** There should be a project existing.

---

**Use Case Name:** CreateAnalysis

**Participating Actors:** Initiated by Project Manager

**Flow of Events:**

1. Project Manager triggers creating an analysis.
2. The system analyzes the repositories in terms of missing links.

**Entry Conditions:** Project Manager enters the analysis list.

**Exit Conditions:** The system saves the analysis of missing links.

**Quality Requirements:** The system should have access to the repositories for the analysis.

---

**Use Case Name:** DeleteAnalysis

**Participating Actors:** Initiated by Project Manager

**Flow of Events:**

1. Project Manager selects an analysis to remove.
2. Project Manager deletes the analysis.
3. The system receives the removal of that analysis.

**Entry Conditions:** Project Manager enters the analysis list.

**Exit Conditions:** The system saves the analysis list without that analysis.

**Quality Requirements:** The system should have the analysis to be deleted.

---

**Use Case Name:** UpdateHistoricalGraph

**Participating Actors:** Initiated by Project Manager

**Flow of Events:**

1. Project Manager inspects the historical graph among pull requests, commits, and issues.
2. Project Manager triggers updating the historical graph.
3. The system fetches new data from the repositories.
4. The system updates the historical graph according to the new data.

**Entry Conditions:** Project Manager enters the historical graph.

**Exit Conditions:** The system saves the historical graph.

**Quality Requirements:** The system should have access to the repositories for the analysis.

---

**Use Case Name:** ManageProjectAccess

**Participating Actors:** Initiated by Project Manager

**Flow of Events:**

1. Project Manager selects a project to manage its access settings.
2. Project Manager selects a person to manage his/her role in that project.
3. Project Manager adds the selected person to the project or removes him/her from the project.
4. The system updates the selected person's access.

**Entry Conditions:** Projects Manager enters the project list.

**Exit Conditions:** The system saves the new status of the selected person for that project.

**Quality Requirements:** A person not in the company cannot be added to the project.

---

**Use Case Name:** RecommendLinks

**Participating Actors:** Communicates with GetRecommendation  
Communicates with GetPossibleMissingLinks

**Flow of Events:**

1. The system runs a ML algorithm to detect possible matches.



2. The system recommends issues found by the algorithm.

**Entry Conditions:** Developer or Project Manager needs link recommendations.

**Exit Conditions:** The system sends the recommended issues.

**Quality Requirements:** The system should recommend at least one issue for each pull request or commit.

---

**Use Case Name:** RetrievePRData

**Participating Actors:** Initiated by Github

Initiated by Azure DevOps

Communicates with RecommendLinks

**Flow of Events:**

1. The system sends a request to GitHub or Azure DevOps to fetch data.
2. GitHub or Azure DevOps gives data requested by the system.
3. The system processes raw data to retrieve pull request data.

**Entry Conditions:** The system needs the pull request data from repositories of a project.

**Exit Conditions:** The system saves pull request data of repositories of a project.

**Quality Requirements:** The system should retrieve pull request data from at least one repository.

---

**Use Case Name:** RetrieveIssueData

**Participating Actors:** Initiated by Github

Initiated by Jira

Initiated by Azure DevOps

Communicates with RecommendLinks

**Flow of Events:**

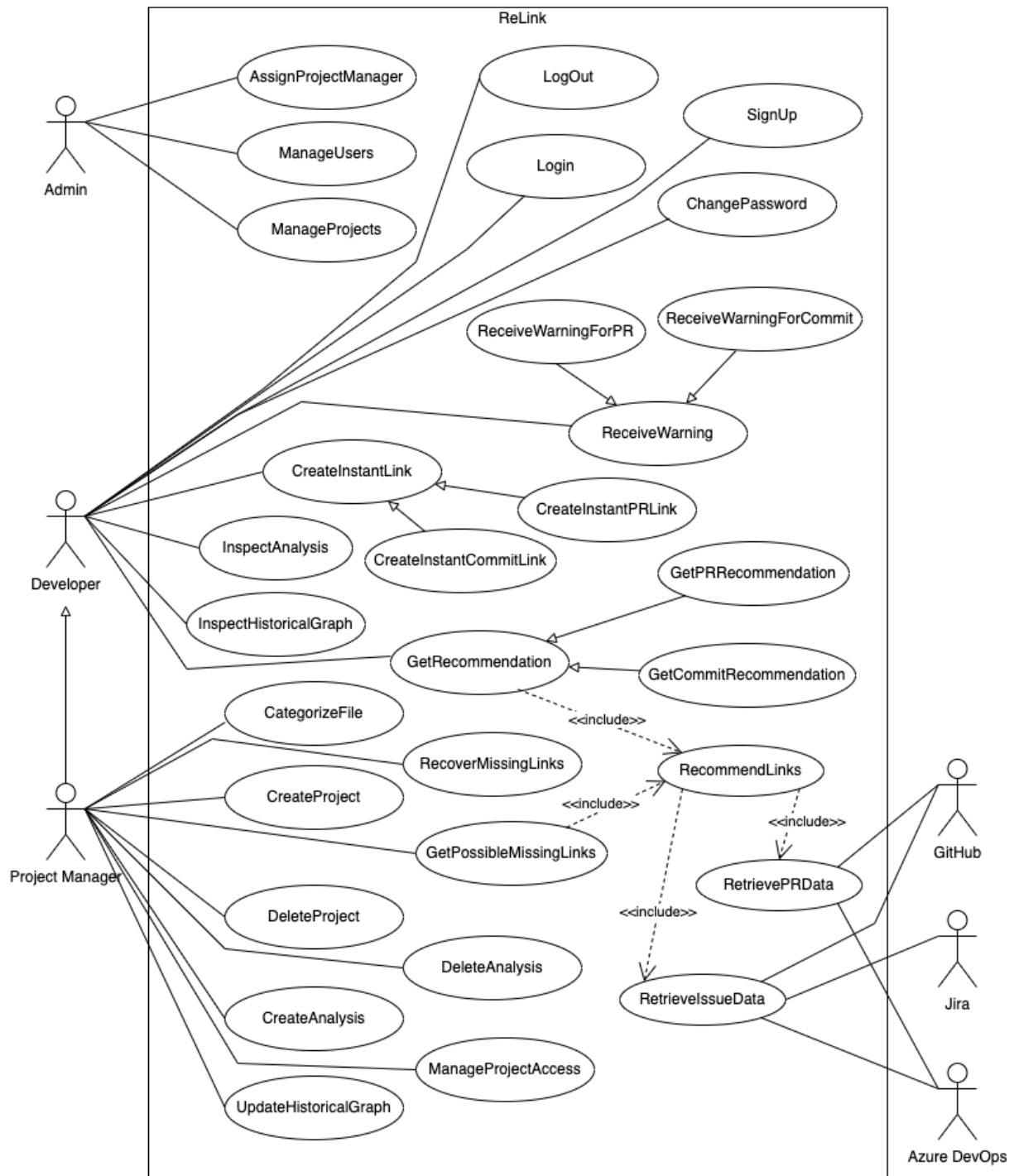
1. The system sends a request to GitHub, Azure DevOps, or Jira to fetch data.
2. GitHub, Azure DevOps, or Jira gives data requested by the system.
3. The system processes raw data to retrieve issue data.

**Entry Conditions:** The system needs the issue data from repositories of a project.

**Exit Conditions:** The system saves issue data of repositories of a project.

**Quality Requirements:** The system should retrieve issue data from at least one repository.

## 2.5.2. Use Case Model



**Figure 1:** Use case diagram.

### 2.5.3. Object and Class Model

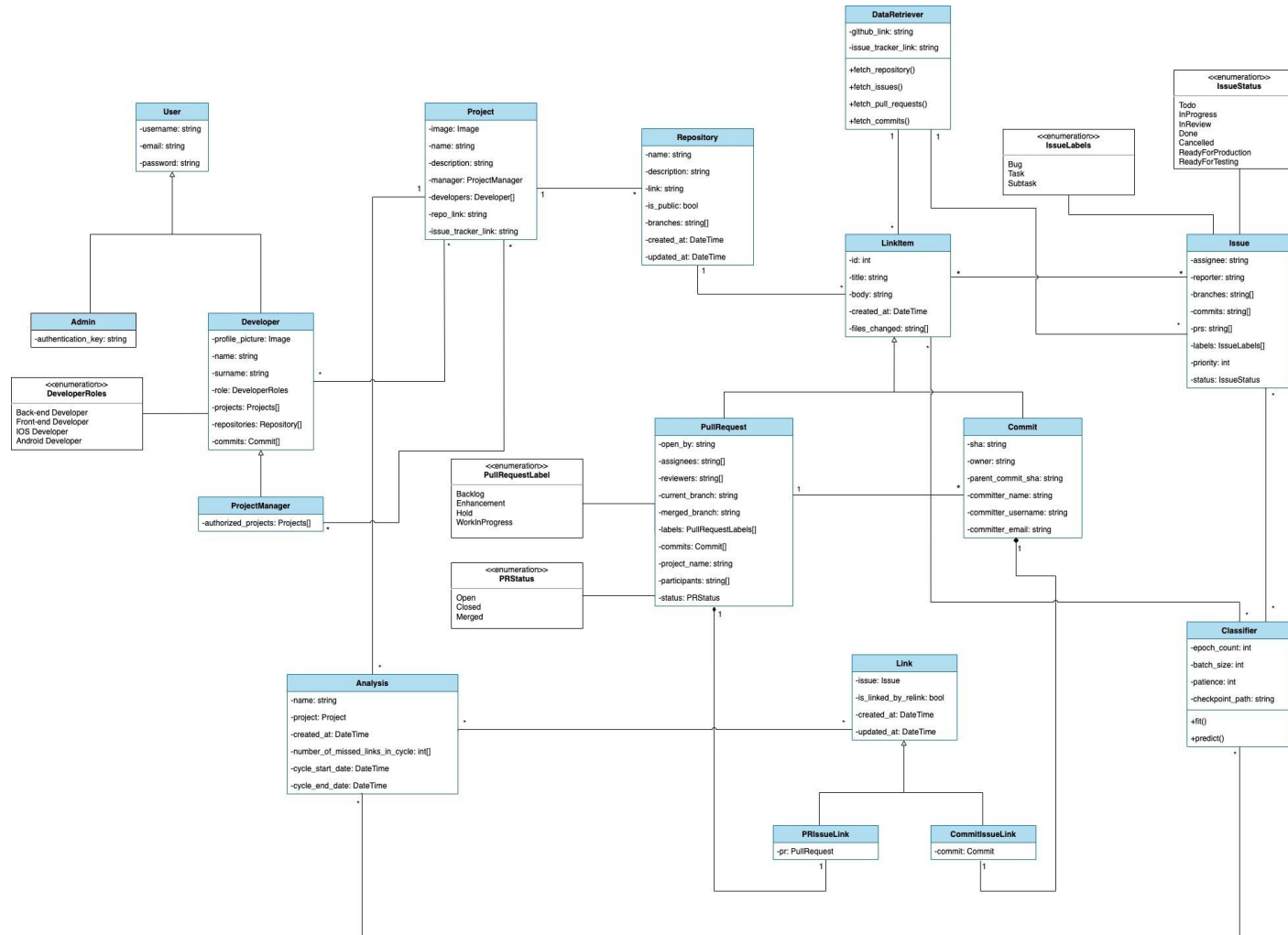


Figure 2: Class diagram.

**User:** Class for generalizing all user types with common attributes.

**Admin:** ReLink employees with full administrative rights.

**Developer:** Developers of a given project with no “write” rights.

**ProjectManager:** Developers with managerial (write) rights on a project.

**DeveloperRoles:** Enumerates integers to developer roles.

**Project:** ReLink project entities.

**Analysis:** An analysis of a given project which gives information on the amount of missed links in a time frame.

**Repository:** Represents repositories associated with a given project in the ReLink system.

**DataRetriever:** Data retrieval class which will collect and format repository information from sources like GitHub and Jira for model training and historical data visualization purposes.

**LinkItem:** Generalizes linkable software artifacts that exist in the ReLink system.

**Commit:** Representation of a GitHub commit on the ReLink system.

**PullRequest:** Representation of a GitHub PR on the ReLink system which will be linked to an issue.

**PullRequestLabel:** Enumerates integers to PR labels.

**PRStatus:** Enumerates integers to PR statuses.

**Issue:** Class for representing issues (such as Jira issues) in ReLink, which are either manually linked, linked by ReLink, has a missing link, or does not require linkage.

**IssueLabels:** Enumerates integers to issue labels.

**IssueStatus:** Enumerates integers to issue statuses.

**Link:** Generalizes the link types ReLink is capable of automatically suggesting.

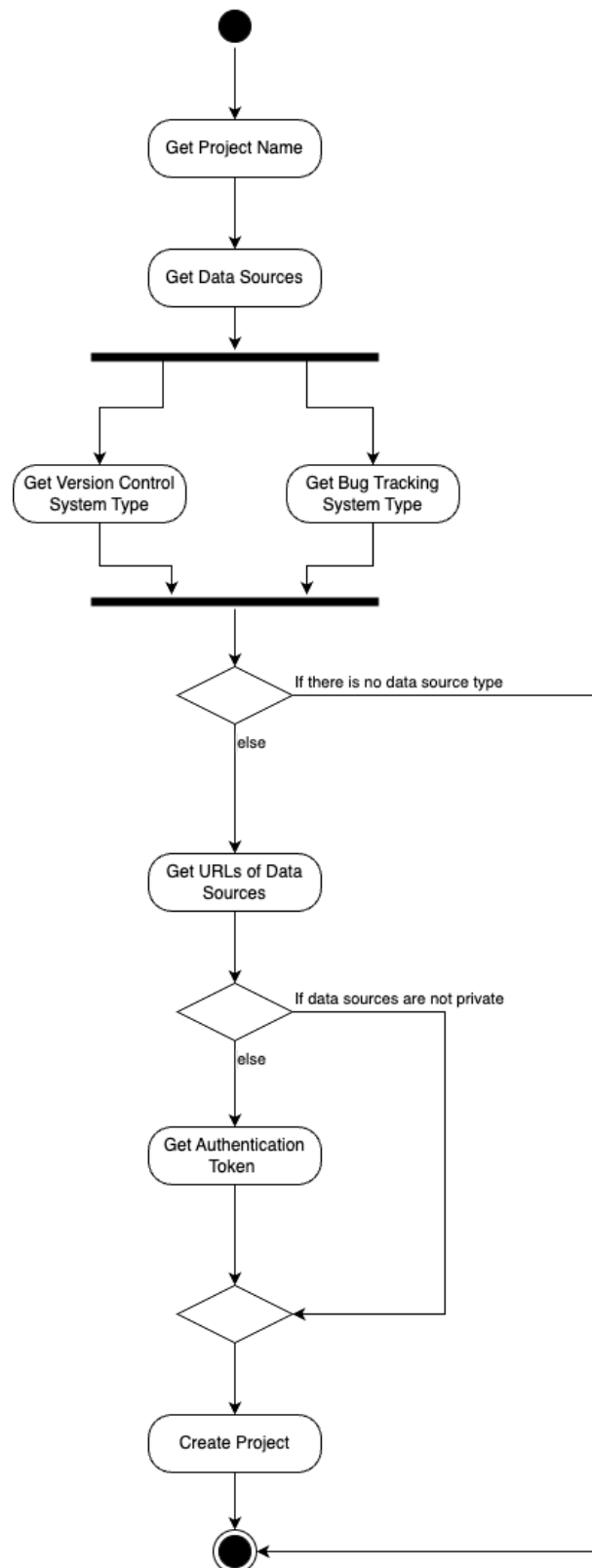
**PRIssueLink:** Class for representing links between PRs and issues.

**CommitIssueLink:** Class for representing links between commits and issues.

**Classifier:** ANN classifier model which will be trained on the data provided by the DataRetriever class. Will be developed using PyTorch.

## 2.5.4. Dynamic Models

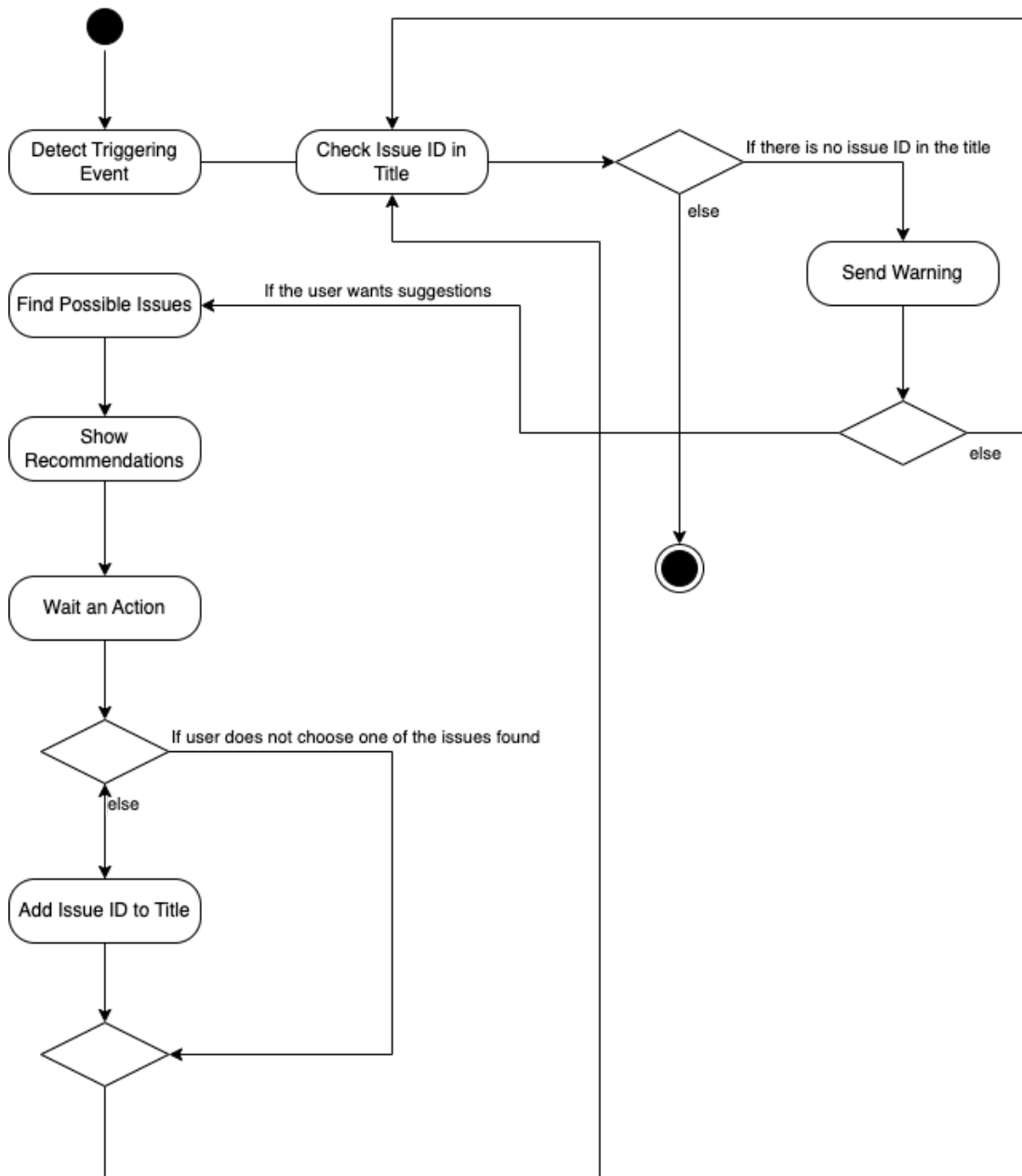
### 2.5.4.1. Create Project Activity Diagram



**Figure 3:** Activity diagram for creating project.

**Explanation:** The system receives the project name and the repositories of the project. If there is no data source, creating project is terminated. If there is, the system receives URLs of each data source. If any of the data sources is not private, creating a project is done. If it is, the system receives an authentication token for each private data source and creates the project.

#### 2.5.4.2. Create Instant Link Activity Diagram

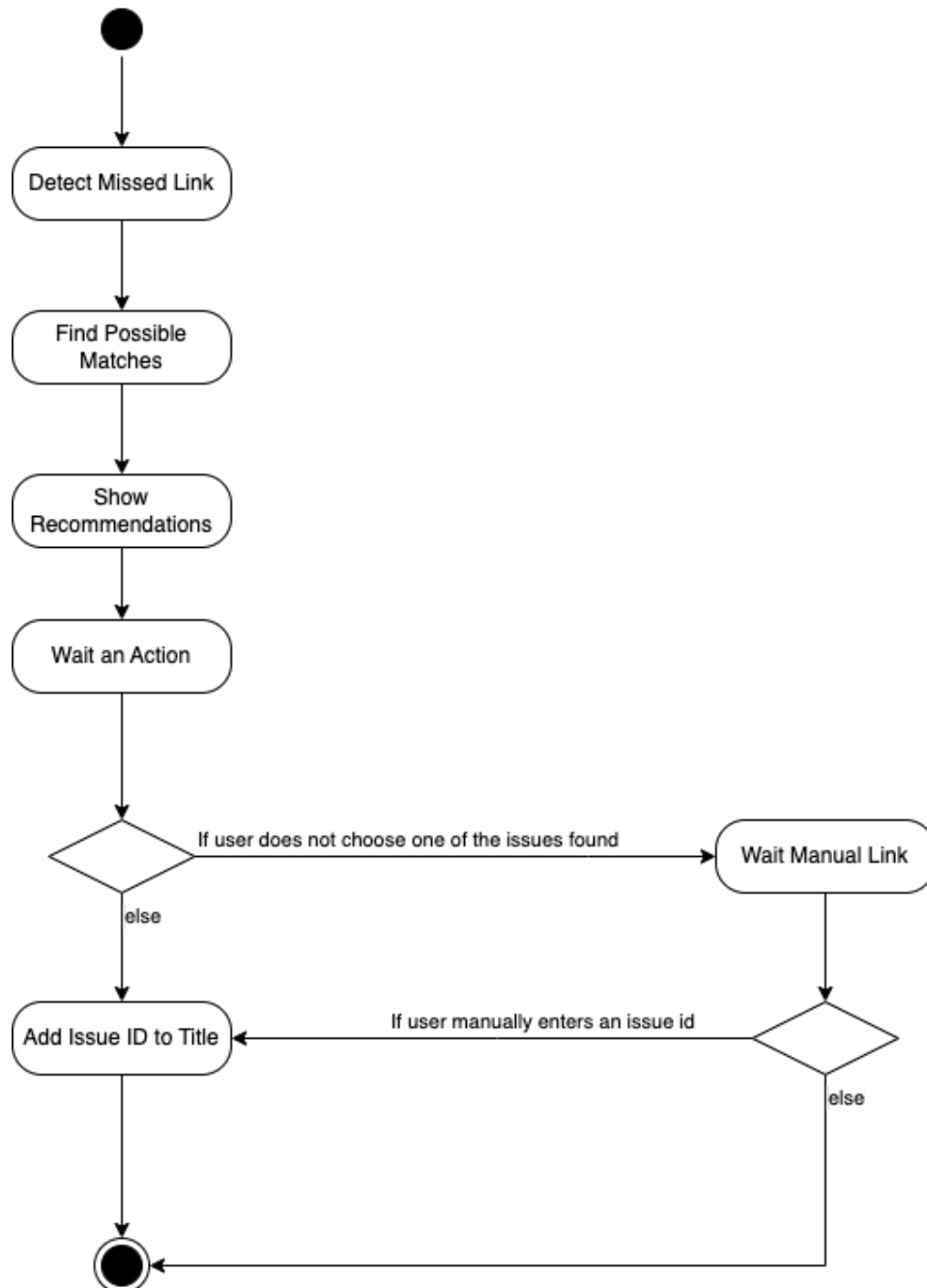


**Figure 4:** Activity diagram for creating instant link.

**Explanation:** The system detects triggering events such as creating a commit or opening a pull request and checks whether an issue id exists in the title. If there is no, the system sends a warning to the user to fix it and recontrol. If there is an issue

id in the title, creating an instant link is not required so it is terminated. If the user wants to get recommendations, the system finds possible matches and provides it to the user. After that, the system waits for an action from the user. If the user chooses an issue, the system adds the id of the selected issue to the title. If the user does not, the system skips adding an issue id to the title. Finally, the system checks again whether an issue id exists in the title. Therefore, the only way to continue to create a commit and open a pull request is to add an issue id to the title.

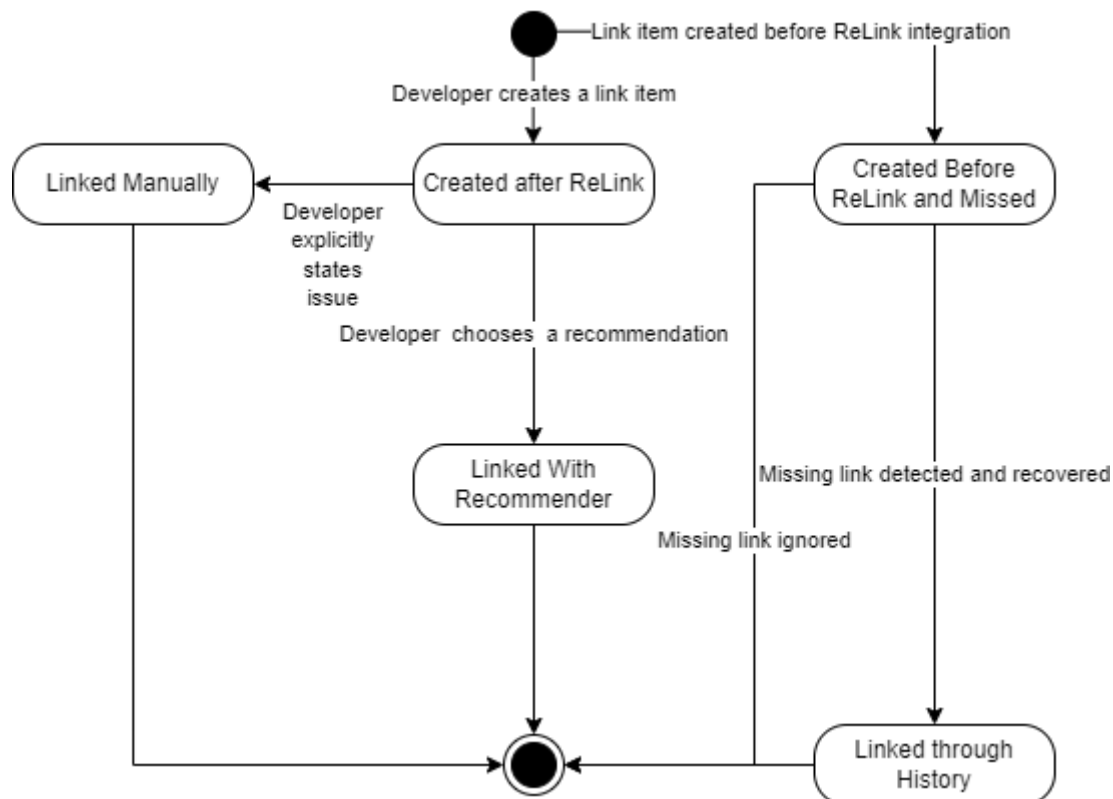
#### 2.5.4.3. Recover Missing Links Activity Diagram



**Figure 5:** Activity diagram for recovering missing links.

**Explanation:** The system detects missing links in the repositories of the project and finds possible matches through a ML algorithm. After that, it shows recommendations found and waits for an action. If the user does not choose one of the recommended issues, the system waits for a manual link from the user. If the user manually enters an issue id or chooses from the issues found, the system adds the selected issue id to the title. If the user neither chooses one of the issues found nor manually enters an issue id, the recovering missing link process is terminated.

#### 2.5.4.4. Link Item State Diagram

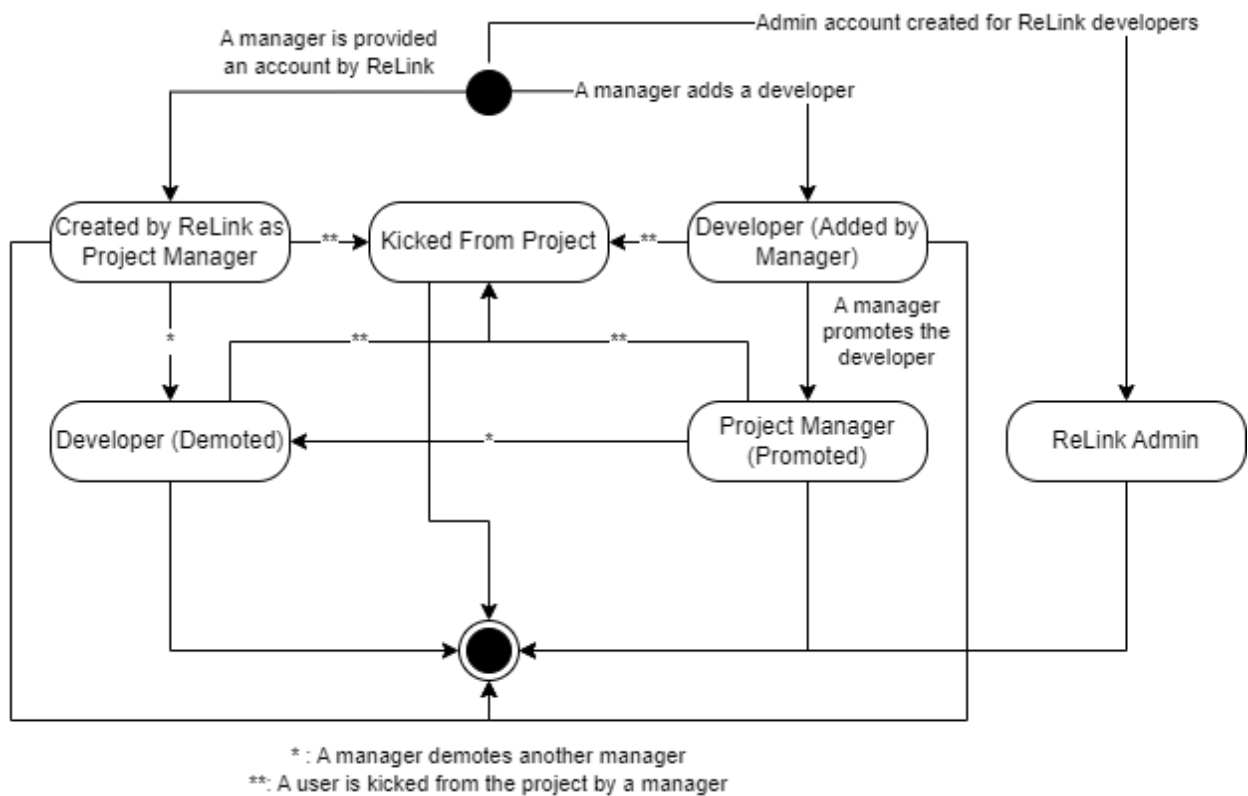


**Figure 6:** State diagram for link items (commits and PRs).

**Explanation:** Here, we observe the states of a link item (either a commit or a PR) throughout stages of linkage, with different linking methods. The state diagrams of commits and PRs are the same, therefore we provide a diagram for their generalization.



### 2.5.4.5. Commit State Diagram

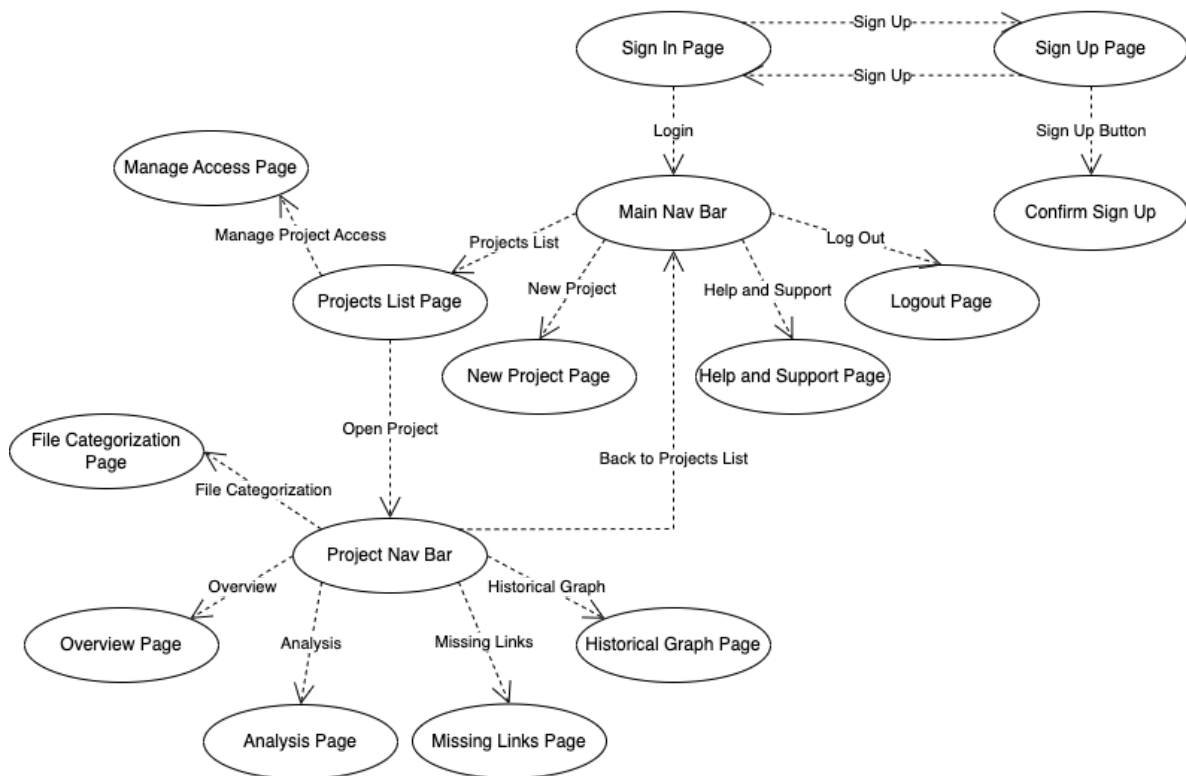


**Figure 7:** State diagram for users.

**Explanation:** Here, we observe the various states of all user types available in ReLink. We see that project managers and developers are dynamic types and can be promoted and demoted while ReLink admin accounts have complete managerial rights and are static.

## 2.5.5. User Interface - Navigational Paths and Screen Mock-ups

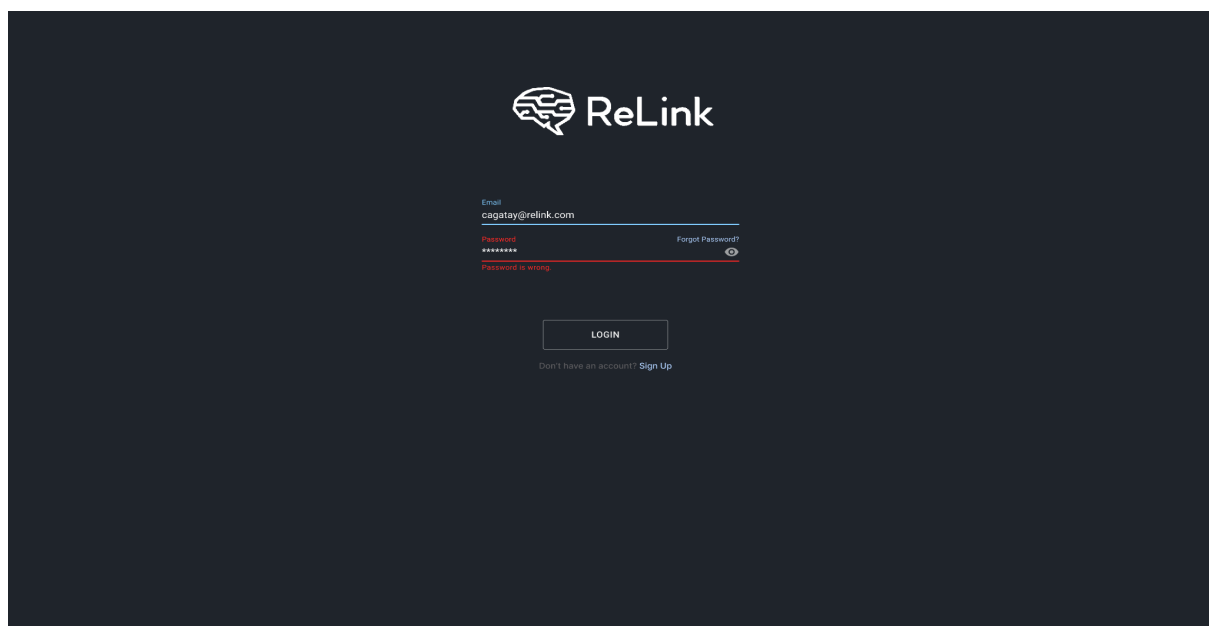
### 2.5.5.1. Navigational Path



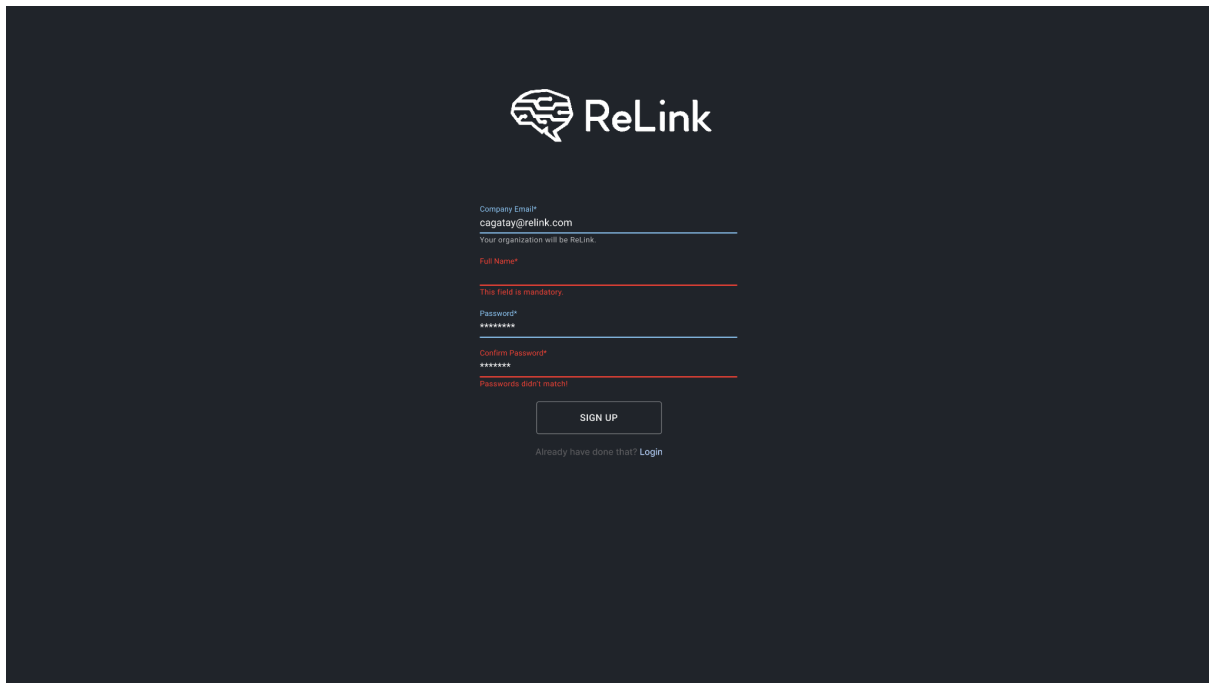
**Figure 8:** Navigational path of the user interface screens.

### 2.5.5.2. Mock-ups

#### 2.5.5.2.1. Login and Register Page



**Figure 9:** Login page.

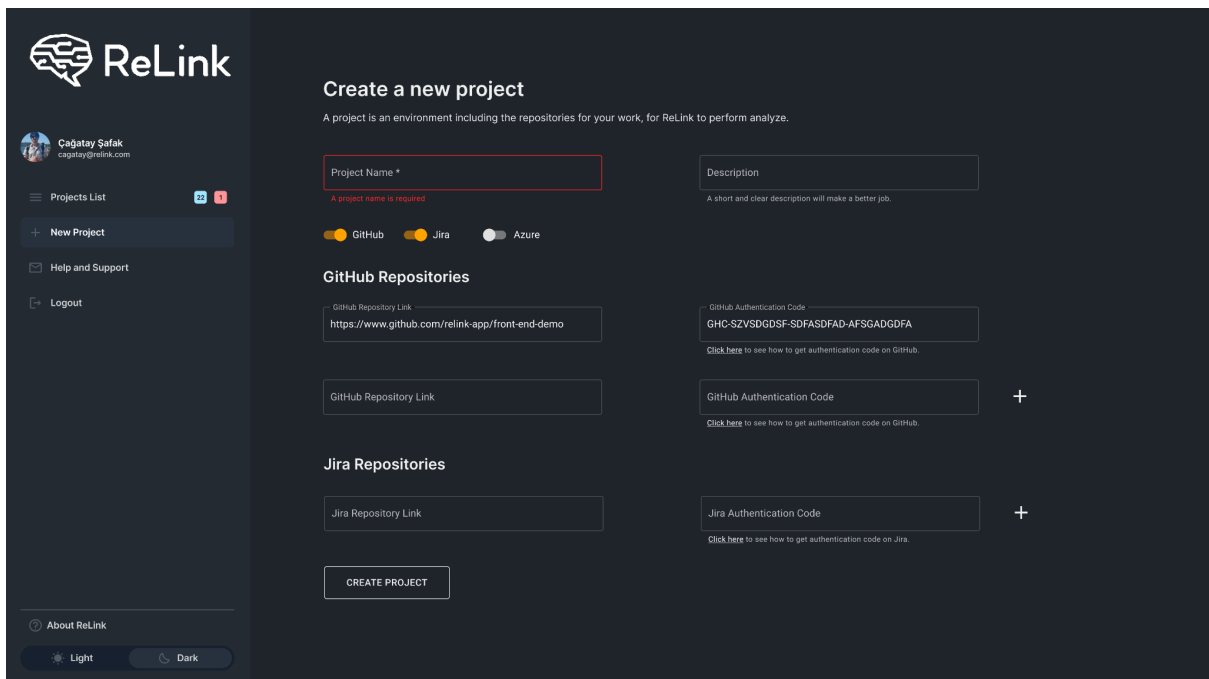


The image shows the ReLink registration page. At the top, there is a ReLink logo consisting of a brain icon with circuit lines and the text "ReLink". Below the logo, the registration form is centered. It includes fields for "Company Email\*" (with the example "cagatay@relink.com" and a note "Your organization will be ReLink"), "Full Name\*" (with a red error message "This field is mandatory"), "Password\*" (with masked characters "\*\*\*\*\*" and a red error message "Passwords didn't match!"), and "Confirm Password\*" (also with masked characters and the same error message). A "SIGN UP" button is located below the password fields. At the bottom, there is a link "Already have done that? Login".

**Figure 10:** Register page.

On these pages developers and project managers can log in to the dashboard. While developers need to sign up to the system before signing in, project managers can sign in with the email and password provided by ReLink to them.

#### 2.5.5.2.2. New Project Page



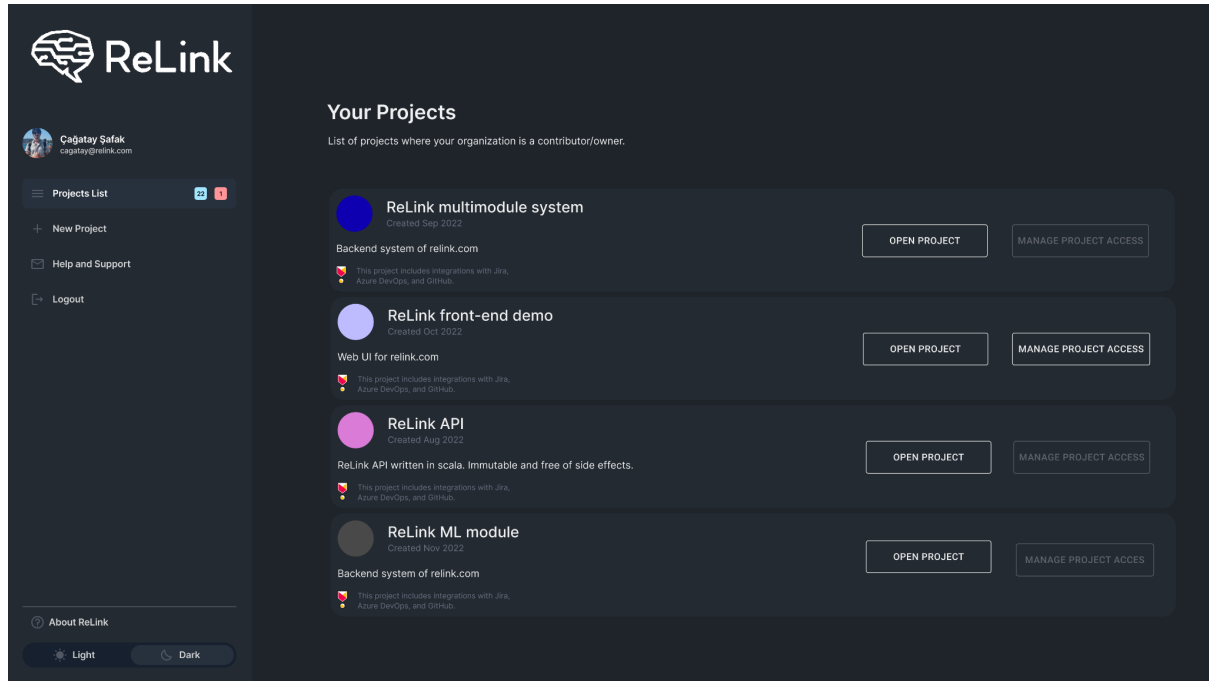
The image shows the ReLink "Create a new project" page. On the left is a sidebar with the ReLink logo, a user profile for "Cağatay Şafak" (cagatay@relink.com), and navigation links: "Projects List", "New Project" (highlighted), "Help and Support", and "Logout". At the bottom of the sidebar are links for "About ReLink" and a theme toggle for "Light" and "Dark". The main content area is titled "Create a new project" and includes a subtitle: "A project is an environment including the repositories for your work, for ReLink to perform analyze." The form contains several fields: "Project Name \*" (with a red error message "A project name is required"), "Description" (with a note "A short and clear description will make a better job."), and radio buttons for selecting the repository type: "GitHub" (selected), "Jira", and "Azure". Below these are sections for "GitHub Repositories" and "Jira Repositories". Each section has a "Repository Link" field and an "Authentication Code" field. The GitHub section shows an example code "GHC-SZVSDGDSF-SDFASDFAD-AFSGADGDFA" and a link to "Click here to see how to get authentication code on GitHub.". The Jira section has a similar structure with a link to "Click here to see how to get authentication code on Jira.". A "CREATE PROJECT" button is at the bottom of the form.

**Figure 11:** New project page.

On this page, the project manager can create a new project. In order to do that, he/she must enter the project name. However, description is optional. Therefore, that

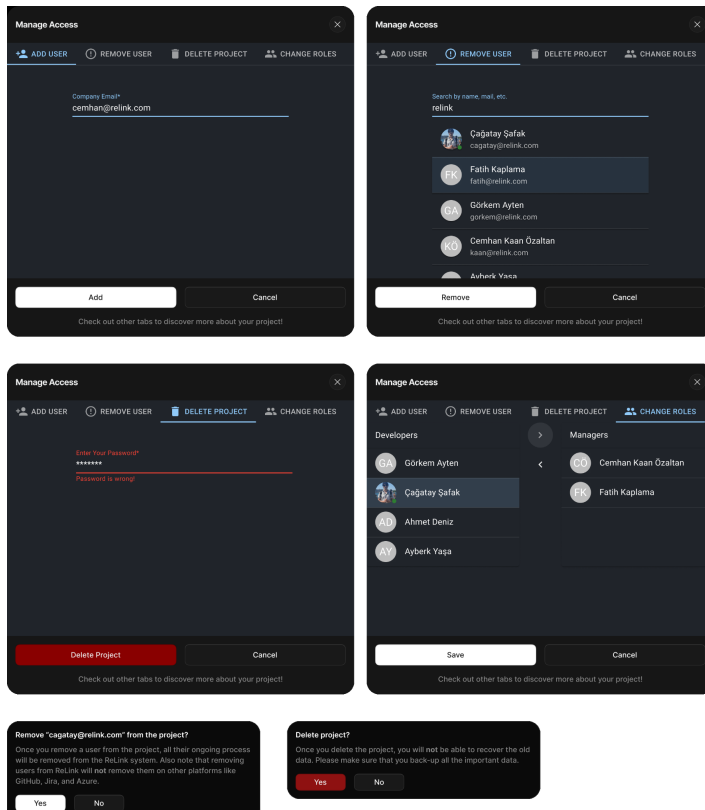
field can be blank. There are 3 toggle buttons for GitHub, Jira, and Azure environments respectively. For each environment, if the user toggles on the button, he/she must fill in the necessary information for repositories such as repository link and authentication code.

### 2.5.5.2.3. Projects Page



**Figure 12:** Projects list page (above).

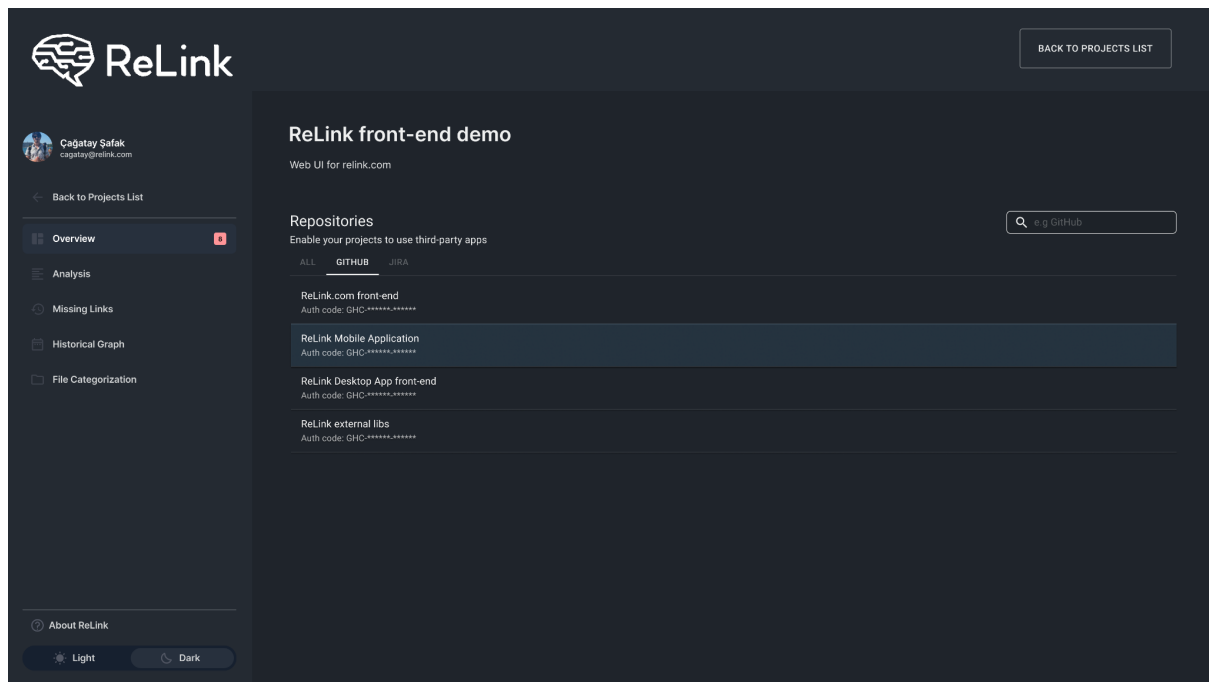
**Figure 13:** Manage project access pop-ups (below).



On this page, developers and the project manager can see the projects list. However, only the project manager has access to the Manage Project Access button.

Each project has information such as description of the project, created date, and which platform includes that project. When the user clicks on the Open Project button, the page below will be shown.

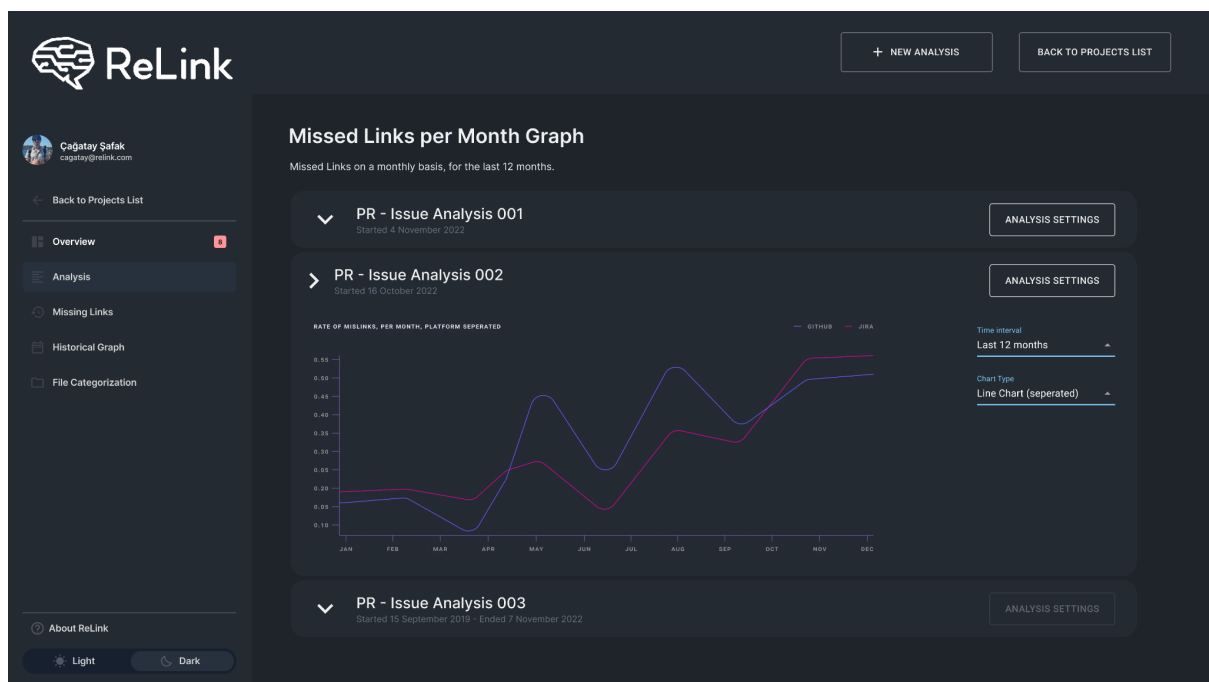
#### 2.5.5.2.4. Overview Page



**Figure 14:** Overview page.

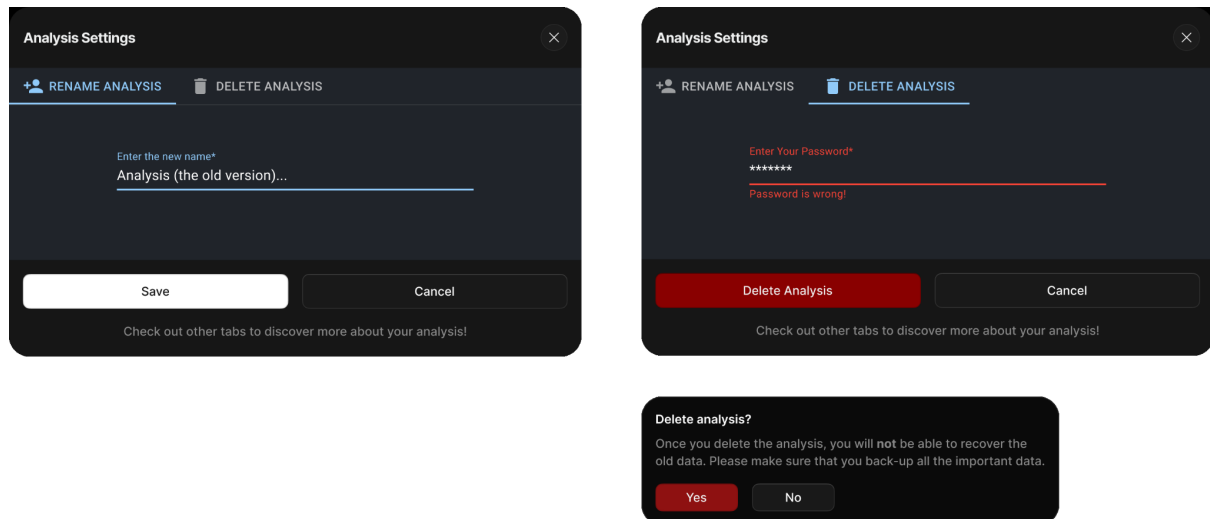
On this page, users can see the general information about the project. This page includes the repositories of the project and some information such as authentication code will be displayed as encrypted. In addition, if the user wants, he/she can go back to the project list page by clicking on the Back to Project List button.

#### 2.5.5.2.5. Analysis Page

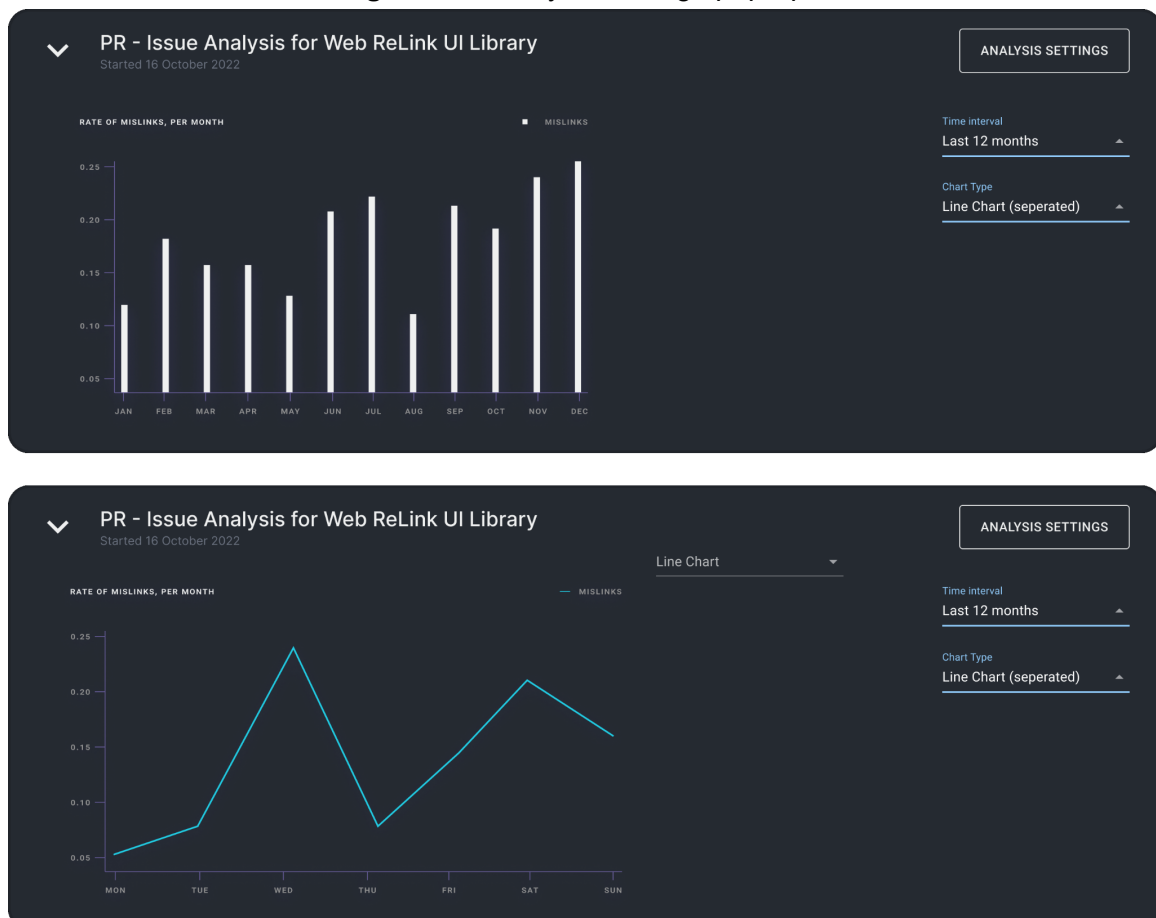


**Figure 15:** Analysis page.

On this page, the user can see the analysis of the project. This page includes the PR - Issue analysis on each card. Each analysis has information of the created date and end date. If the user wants, he/she can get a new analysis of the project by clicking on the New Analysis button on the top bar. Each card includes the visual graph of the analysis on a monthly basis. The user can change the time interval and chart type by clicking on the dropdown menu.

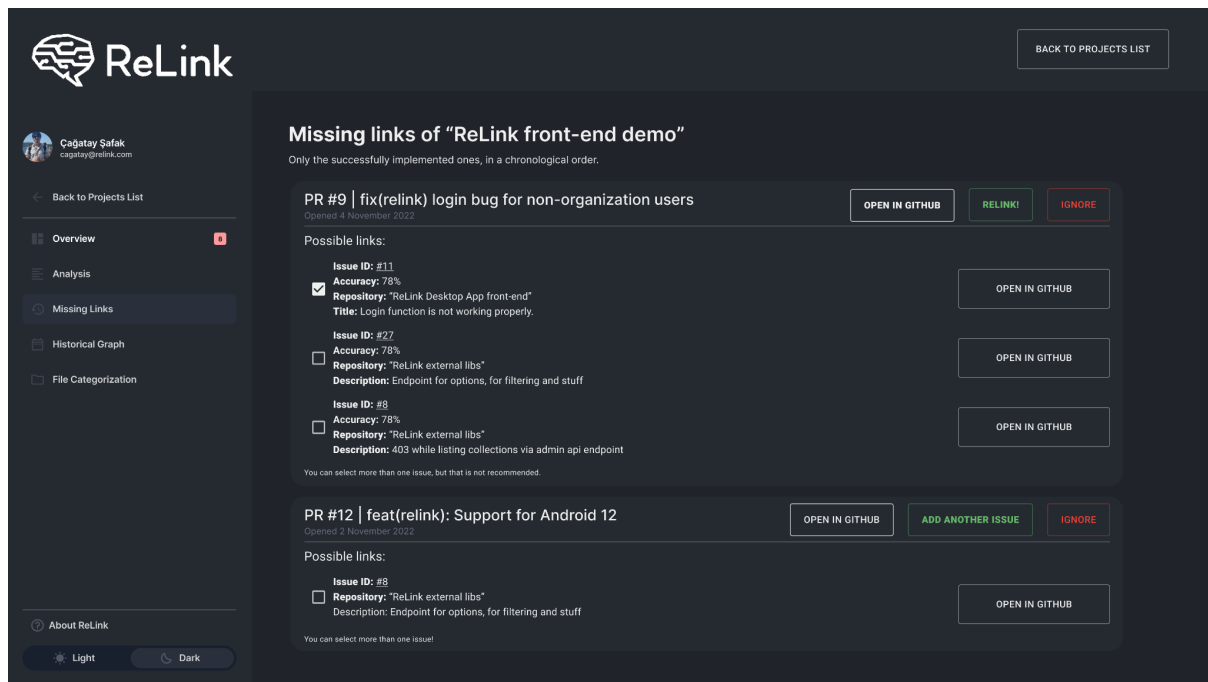


**Figure 16:** Analysis settings pop-ups.



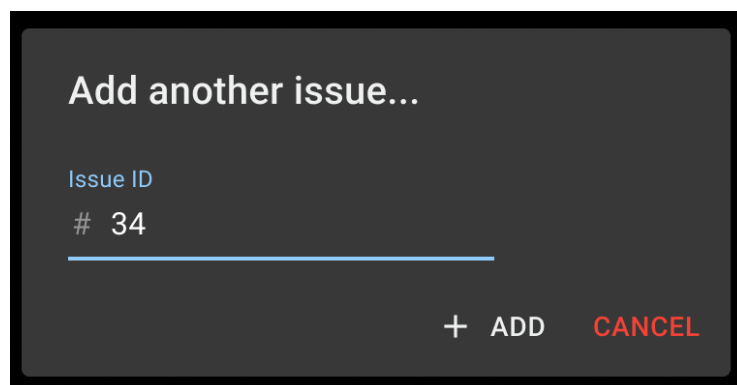
**Figure 17:** Different analysis chart types.

### 2.5.5.2.6. Missing Links Page



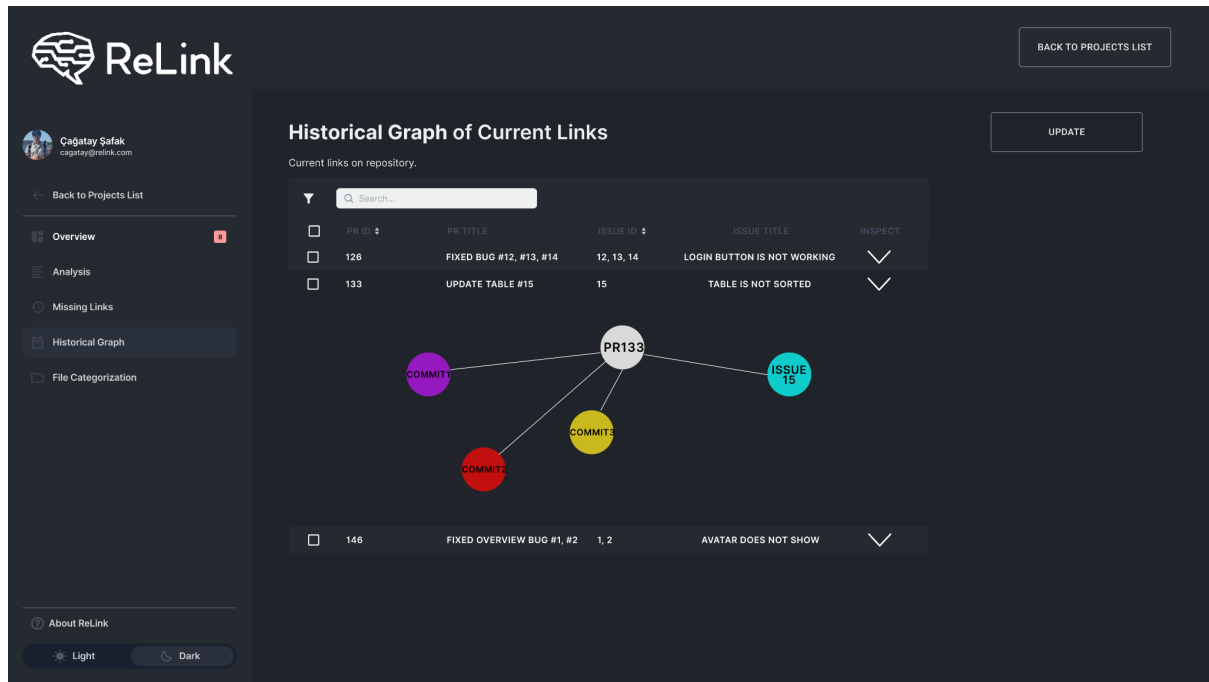
**Figure 18:** Missing links page.

On this page, the user can see the possible PR - Issue matches which are not linked. For each PRs possible links will be displayed to the user and the user can select one of them. If no issue is related to PR, the user can skip that PR by clicking on the ignore button. If the user thinks that one of the possible issues recommended is related to the PR, he/she can link the PR and issue by clicking on the Relink button. Thanks to this page, the user can link the past missing links.



**Figure 19:** Add another issue than the recommended ones pop-up.

### 2.5.5.2.7. Historical View Page

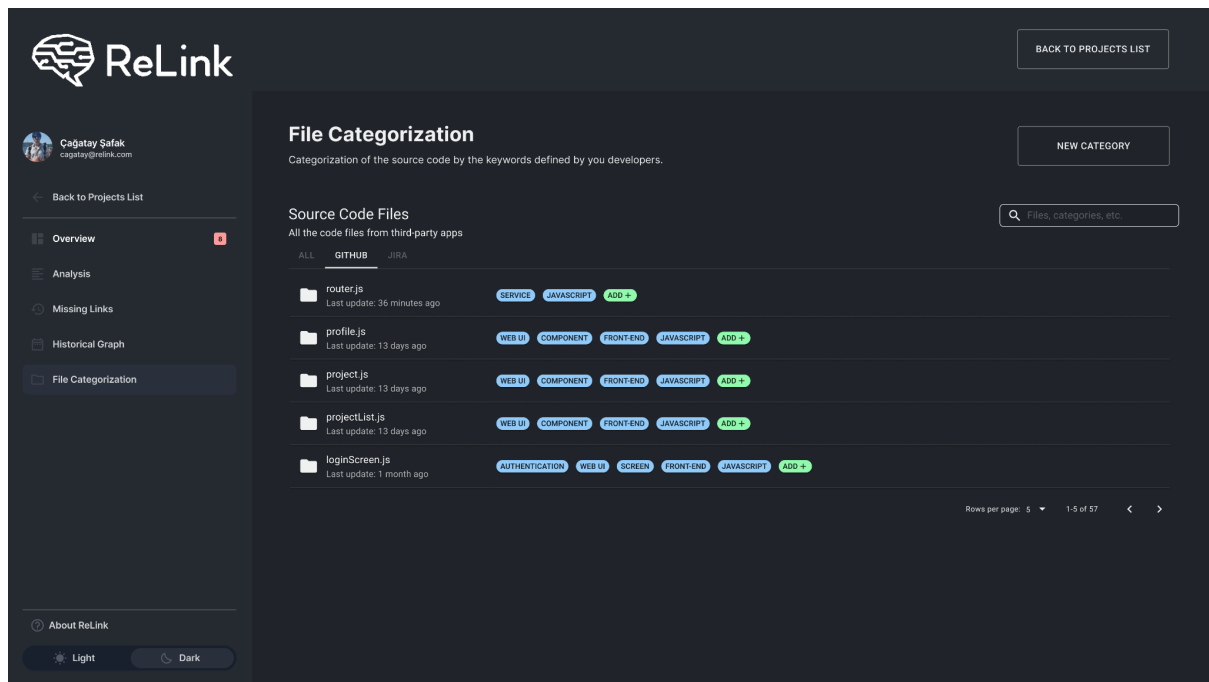


**Figure 20:** Historical view page.

On this page, the user can see the historical view of the current links. This information is displayed in a datatable. Each row contains the information such as PR id, PR title, issue id, issue title. Besides, if the user clicks on the inspect button, he/she can see the visual graph of the current links. Each commit and issue is displayed connected to PR. Users can search the links or sort them. In addition, when the user clicks on the update button, if there is a new link on the project, it is displayed in the datatable.

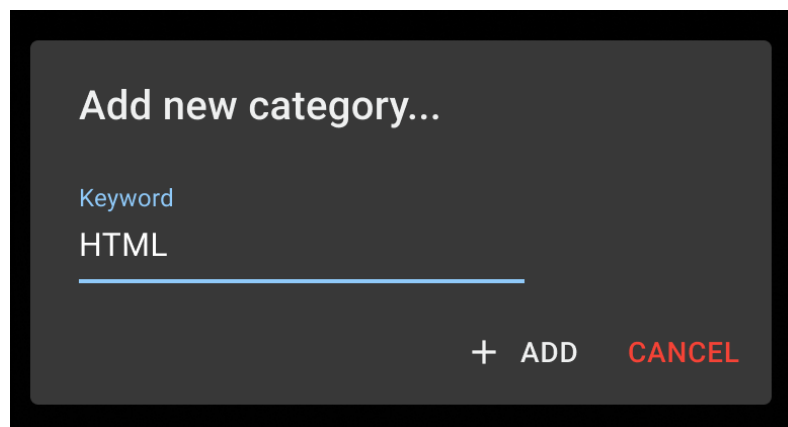


### 2.5.5.2.8. File Categorization Page



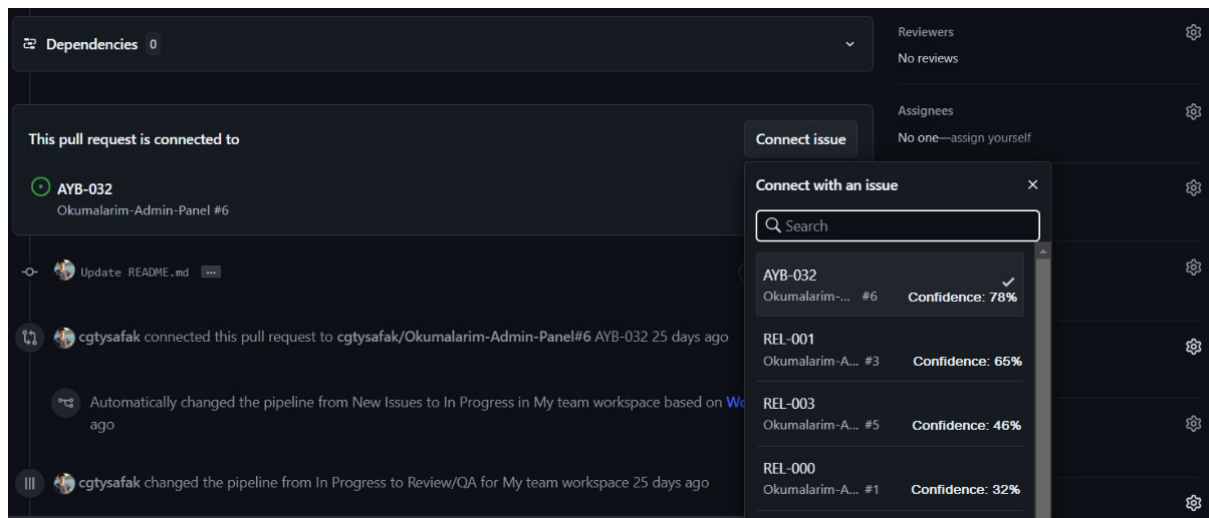
**Figure 21:** File categorization page.

On this page, the user can see each of the source files on the project. Users should be able to label these files however they want. ReLink will use those data to provide a better PR - Issue matching for those files. These labels when training ML model.



**Figure 22:** Add a new type of categorization pop-up.

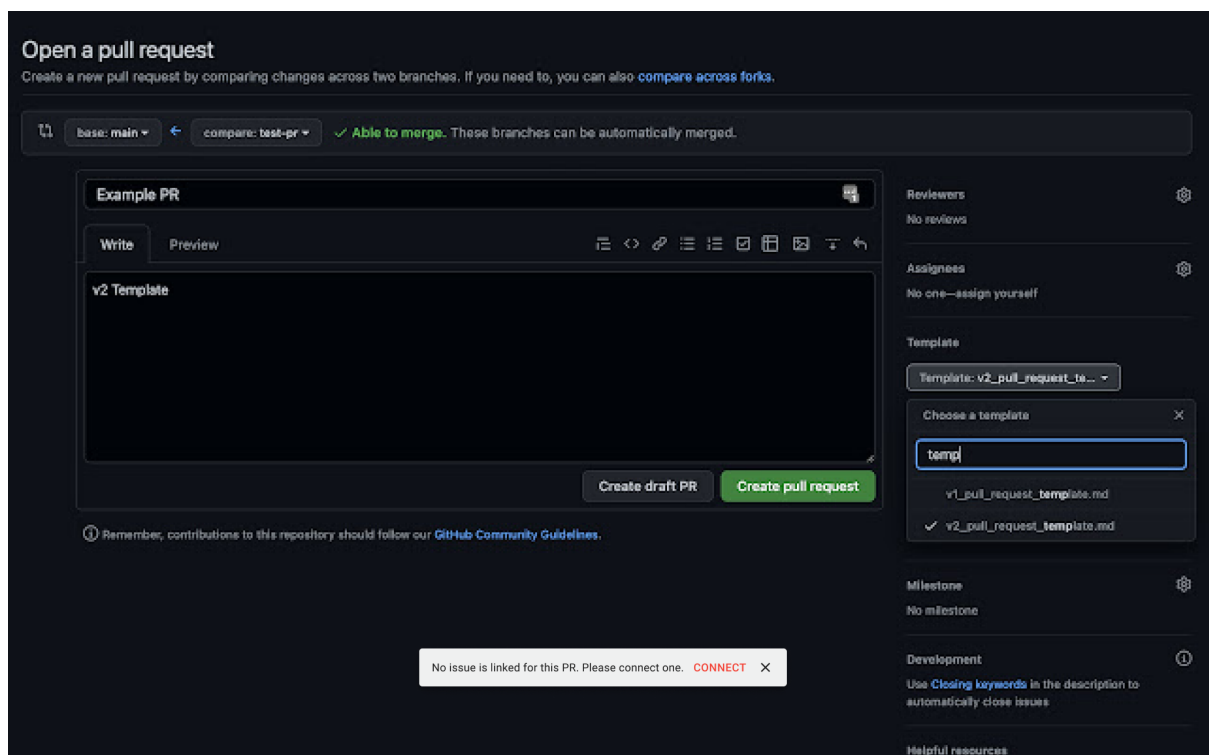
### 2.5.5.2.9. Create Instant Link Page



**Figure 23:** Create an instant link page.

On this page, the user can create an instant link, i.e., the user can add an issue link when creating a pull request looking at sorted confidence predictions. This page is embedded in the GitHub Web UI. The same interface is available for commits.

### 2.5.5.2.10. Warning Page



**Figure 24:** A demo of the 'missing link' warning toast.

On this page, the user receives a warning message with a toast component when the user tries to create a pull request without an issue link.

## **3. Other Analysis Elements**

### **3.1. Consideration of Various Factors in Engineering Design**

There is no external factor except for the economic factor limiting us during the development of ReLink. We need to consider some economic factors during the design phase of the project. These economic factors are database costs and server costs, respectively. Since we will be doing repository mining, we need a database where we can store the data of the project with any scale from GitHub, Jira, or Azure DevOps. This database will require a substantial budget. Moreover, we will need external GPUs to train the Model, so we will need more performance than a normal web application. This will increase server costs.

### **3.2. Risks and Alternatives**

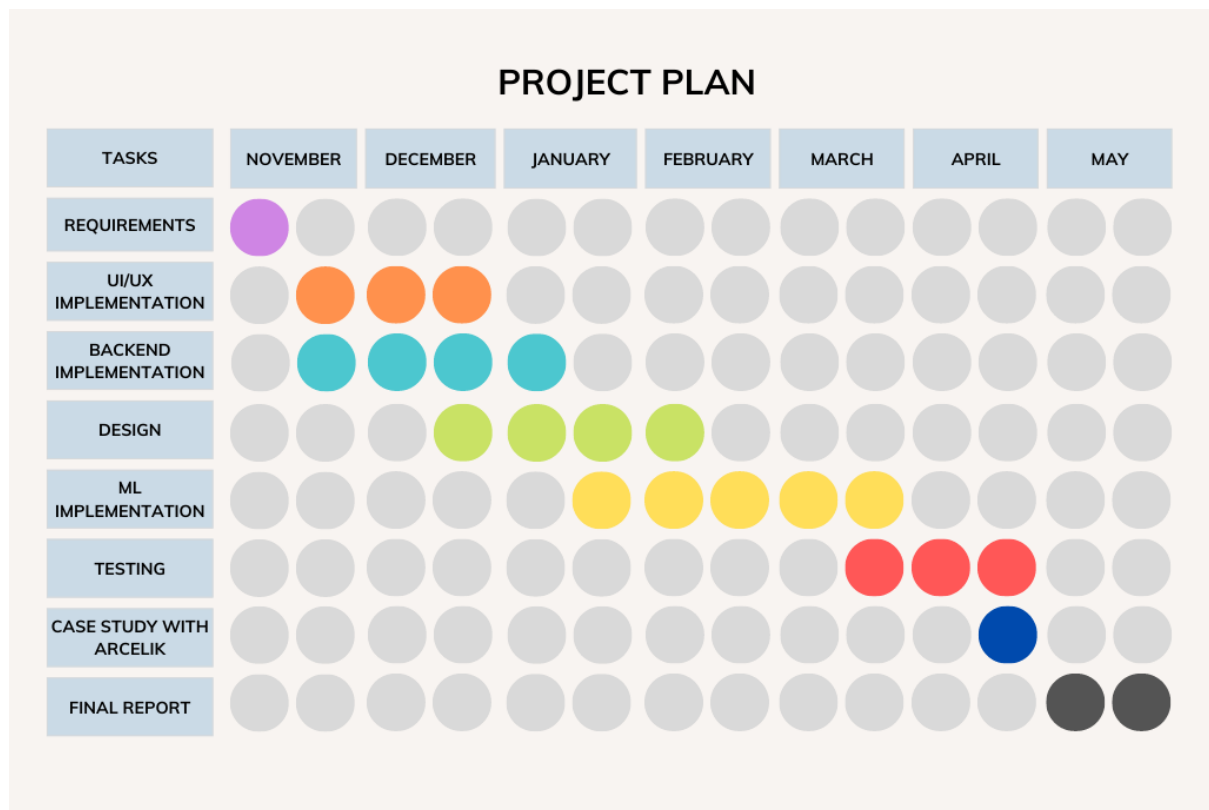
The most prevalent risk for the development of ReLink is the values of the prediction model's metrics, i.e. accuracy, precision, recall, and F1 score. Other than these metrics, the development of this project is relatively straightforward as the rest of the functionalities are standard UI features. For our prediction model, we will be using an artificial neural network (ANN) model developed with PyTorch and extract our features from the metadata of the commits and pull requests of existing projects. The classes of the N most active neurons in the output layer will then be recommended to the user during the commit or PR stage, where the classes will be the active issues in our case. Since our model will fit on a relatively unique kind of data, we cannot be absolutely certain that we will be able to deliver a model with high accuracy (such as in the range of 80% to 99%). In addition, even though we will be able to deliver a model with an acceptable accuracy, and deep learning methods proved to be more useful than traditional machine learning models [11], we may still get results with low accuracy and high variance [12]. For example, in the natural language processing domain, attempting to classify complex statements may result with low accuracy values as demonstrated by Robeer et al. [13]. This may also be the case as we are attempting to make classifications based on the metadata and developer messages which may be similar in complexity to such cases. In addition, other common reasons for low accuracy results may be class imbalance where the training data is skewed towards a class, or overfitting of the model to the data [14]. In the face of these risks, even though several studies exist arguing that a model with this function that has high accuracy can be developed [15, 16, 17], we may end up with a very low accuracy (such as 10%). In this case, we will be pivoting to alternative directions in our project. For example, we can increase the number of recommendations made for the developer (N) in the case of a low accuracy which is still higher than the chance level. In the worst case scenario where our accuracy is no different than the chance level, the most suitable alternative would be to pivot to a

project that enforces developers to make explicit links while creating commits and PRs, in addition to providing visualization about their historical progress. Granted this reduces the scope of the project significantly, it also means that our project still has a feasible alternative direction even in the worst possible case.

Another risk is the fact that the models used in this project will be project specific, as the issues in each project will be unique. This means that a unique model will have to be trained for each project, with the previous commits and PRs of the respective project being used for training. In addition, as new explicit links are created, they will serve as new labeled data, and the model will be trained further to increase performance and adapt to changes in the project. All of this means that the training process may be computationally expensive, and therefore affect our performance. An alternative solution to this problem could be to reduce the re-training frequency if our performance is affected during development. Another alternative could be to determine issue types that can be used across different projects, therefore reducing the need for a unique model for each project to an extent.

Another risk would be in the field of data collection for model training. Since we will be using private repositories in addition to public ones to obtain the maximum amount of data, the access level we are given to private repositories by companies is crucial. More importantly, we will be using APIs provided to us for data retrieval, which creates a compatibility risk. For example, the data fields supported by the provided API may be insufficient and may not include the data required to extract the most important features for our model. A possible alternative to this risk could be to ask for elevated permissions, or to simply adapt our model in a way in which it can utilize the provided format to its fullest.

### 3.3. Project Plan



**Figure 25:** Project plan represented with gantt chart.

### 3.4. Ensuring Proper Teamwork

Besides asynchronous communication through Google Drive to cooperate on reports, there are regular meetings such as face to face and online weekly meetings within the team. If any topics are discussed or any decision related to analysis, design and implementation is taken, these are recorded in the cloud service providers. Besides these meetings within the team, we have biweekly meetings with Eray Tüzün who is the supervisor of the project ReLink. Since we have decided to follow the principles of an Agile Project Management Methodology, specifically Scrum, we have adjusted our sprint length to 2 weeks to coincide with the meetings with Eray Tüzün. Therefore, we have to perform the sprint review, retrospective and planning in the same meeting under the supervision of Eray Tüzün. To make it easier to follow sprints, we have decided to use the Scrum board in Jira which is the project management and issue tracking system. In this way, tasks are selected from the backlog at the beginning of each sprint for each member of the team in accordance with each member's workload related to other lectures, and everyone's progress is tracked throughout the sprint. For example, if anyone waits for a pull request review from other team members, it's immediately helped.

### 3.5. Ethics and Professional Responsibilities

We give importance to privacy by design which is a system design that takes into account the data protection at the beginning of the design and architecture of the application [12]. While ReLink mines the repositories of the open source projects, it has to access the commit, pull request, and issue data of the projects on GitHub, Azure DevOps, and Jira. ReLink uses mining data to match pull requests or commits and issues. However, while doing this, we will use encryption algorithms like SHA256 to prevent any data leaks and protect the data of the projects. Moreover, we will ensure that we will only read data and never overwrite data while mining from version control systems and bug tracking systems, from which we will obtain project-related data.

### 3.6. Planning for New Knowledge and Learning Strategies

Everyone on our team had experience in web application development. Some of us had experience as front-end developers, some of us as back-end developers, and some of us as full stack developers. But no one in the team had developed an ML-based application before. That's why we read conference and journal articles written on this topic [15, 16, 17]. We reviewed ML-based tools and algorithms made for articles. Since we do not have ML experience, we watched tutorial series [18] on YouTube about it. To be more specific, we watched videos about Deep Learning and PyTorch.

## 4. Glossary

**Issue-PR / Issue-Commit Link:** Semantic associations created between the given software artifacts, either manually or by the ReLink system.

**Commit:** A change on a file, or a set of files, in a version control system repository.

**Pull Request (PR):** The event of merging new code to the code repository in software development.

**Issue:** Descriptions of needed changes and tasks on a software project that allow tracking progress.

**Traceability:** The capacity to explicitly relate associated software artifacts throughout development [16].

## 5. References

- [1] T. W. Aung, H. Huo, and Y. Sui, "A literature review of automatic traceability links recovery for software change impact analysis," *Proceedings of the 28th International Conference on Program Comprehension*, 2020. [Online]. Available: <https://doi.org/10.1145/3387904.3389251>. [Accessed: 29-Sep-2022].
- [2] M. Rath, J. Rendall, J. L. Guo, J. Cleland-Huang, and P. Mäder, "Traceability in the wild," *Proceedings of the 40th International Conference on Software Engineering*, 2018. [Online]. Available: <https://doi.org/10.1145/3180155.3180207>. [Accessed: 29-Sep-2022].
- [3] C. M. Lüders, T. Pietz, W. Maalej. "Automated Detection of Typed Links in Issue Trackers". [Online]. Available: <https://doi.org/10.48550/arXiv.2206.07182>. [Accessed: 29-Sep-2022].
- [4] "Web Content Accessibility Guidelines (WCAG) 2.1". <https://www.w3.org/TR/WCAG21/>. [Accessed: Oct 23, 2021].
- [5] "Django". <https://www.djangoproject.com/>. [Accessed: Oct 23, 2021].
- [6] "React – a JavaScript library for building user interfaces," *React*. [Online]. Available: <https://reactjs.org/>. [Accessed: 08-Oct-2022].
- [7] "High performance, kubernetes native object storage," *MinIO*. [Online]. Available: <https://min.io/>. [Accessed: 08-Oct-2022].
- [8] "The developer Data Platform," *MongoDB*. [Online]. Available: <https://www.mongodb.com/>. [Accessed: 08-Oct-2022].
- [9] "Pytorch," *PyTorch*. [Online]. Available: <https://pytorch.org/>. [Accessed: 08-Oct-2022].
- [10] "Getting started with the built-in Bert Algorithm," *Google*. [Online]. Available: <https://cloud.google.com/ai-platform/training/docs/algorithms/bert-start>. [Accessed: 08-Oct-2022].
- [11] C. Janiesch, P. Zschech, and K. Heinrich, "Machine learning and deep learning," *Electronic Markets*, vol. 31, no. 3, pp. 685–695, 2021. [Online]. Available: <https://doi.org/10.1007/s12525-021-00475-2>. [Accessed: 08-Oct-2022].
- [12] S. Liu, Y. Wang, X. Yang, B. Lei, L. Liu, S. X. Li, D. Ni, and T. Wang, "Deep Learning in medical ultrasound analysis: A Review," *Engineering*, vol. 5, no. 2, pp. 261–275, 2019. [Online]. Available: <https://doi.org/10.1016/j.eng.2018.11.020>. [Accessed: 08-Oct-2022].
- [13] M. Robeer, G. Lucassen, J. M. van der Werf, F. Dalpiaz, and S. Brinkkemper, "Automated extraction of conceptual models from user stories via NLP," *2016 IEEE 24th International Requirements Engineering Conference (RE)*, 2016. [Online]. Available: <https://doi.org/10.1109/RE.2016.40>. [Accessed: 08-Oct-2022].
- [14] "9 reasons why machine learning models not perform well in production," *Toward Data Science*. [Online]. Available: <https://towardsdatascience.com/9-reasons-why-machine-learning-models-not-perform-well-in-production-4497d3e3e7a5>. [Accessed: 10-Nov-2022].
- [15] T. W. Aung, H. Huo, and Y. Sui, "A literature review of automatic traceability links recovery for software change impact analysis," *Proceedings of the 28th International Conference on Program Comprehension*, 2020. [Online].

- Available: <https://doi.org/10.1145/3387904.3389251>. [Accessed: 29-Sep-2022].
- [16] M. Rath, J. Rendall, J. L. Guo, J. Cleland-Huang, and P. Mäder, "Traceability in the wild," *Proceedings of the 40th International Conference on Software Engineering*, 2018. [Online]. Available: <https://doi.org/10.1145/3180155.3180207>. [Accessed: 29-Sep-2022].
  - [17] C. M. Lüders, T. Pietz, W. Maalej. "Automated Detection of Typed Links in Issue Trackers". [Online]. Available: <https://doi.org/10.48550/arXiv.2206.07182>. [Accessed: 29-Sep-2022].
  - [18] "Deep learning with PyTorch - full course," *YouTube*, 24-Feb-2021. [Online]. Available: <https://youtu.be/c36lUUr864M>. [Accessed: 11-Nov-2022].