FACULTY
OF INFORMATION
TECHNOLOGY
CTU IN PRAGUE

Assignment of bachelor's thesis

Title:	A module for detecting plagiarism in LearnShell
Student:	Zbyněk Juřica
Supervisor:	Ing. Jakub Žitný
Study program:	Informatics
Branch / specialization:	Web and Software Engineering, specialization Software Engineering
Department:	Department of Software Engineering
Validity:	until the end of summer semester 2021/2022

Instructions

LearnShell is a modular system for managing and performing exams with programming assignments in scripting languages, especially Shell. LearnShell currently offers basic functionality for detecting duplicate submissions (when there is a 100% match).

There is a space for improvement in the front-end and in the detection algorithm itself.
1. Analyze the current state of the solution and propose high-level improvements for the front-end and algorithm.
2. Focus on the UX of the front-end application and propose a solution that will enable teachers to:
- inspect and compare duplicate submissions of students,
- aggregate groups of students with duplicated submissions,
- resolve plagiarism suspicions,
- export data.
3. Implement basic improvements in the duplicate detection algorithm.

Electronically approved by Ing. Michal Valenta, Ph.D. on 18 January 2021 in Prague.

FACULTY
OF INFORMATION
TECHNOLOGY
CTU IN PRAGUE

Bachelor's thesis

A module for detecting plagiarism in LearnShell

Zbyněk Juřica

Department of Software Engineering
Supervisor: Ing. Jakub Žitný

May 13, 2021

Acknowledgements

I would like to thank my supervisor Ing. Jakub Žitný, for the patience, guidance and all the helpful objections and advice that he has given to me. I am also grateful to my family for supporting me and helping with anything I needed, despite of their busy schedules.

Declaration

Citation of this thesis

Juřica, Zbyněk. *A module for detecting plagiarism in LearnShell.* Bachelor's thesis. Czech Technical University in Prague, Faculty of Information Technology, 2021.

Abstrakt

Tato bakalářská práce ze zaměřuje na vylepšení stávajícího modulu pro detekci plagiátorství v LearnShellu. Cílem této práce je vytvořit front-end React aplikaci s důrazem na uživatelský prožitek, která pomůže učitelům zobrazovat podezření na plagiátorství a jejich následné řešení. Další část této práce se zaobírá vylepšením samotného algoritmu pro detekci plagiátů. Tato práce popisuje celý proces takovéhoto vývoje od analýzy a praktických příkladů a popisu implementace všech stanovených cílů, tak i další teoretické možnosti pro budoucí vylepšení.

Klíčová slova Detekce plagiátorství, Django, Python, React, GraphQL, TypeScript, Next.js, JavaScript, Vizualizace dat, Abstraktní syntaktický strom

Abstract

This bachelor's thesis focuses on improving the current module for detecting plagiarism in LearnShell. The goal is to build a front-end React application with an emphasis on user experience to allow teachers to inspect the plagiarism suspicions and resolve them. Another part of this work aims to improve the plagiarism detection algorithm itself. This work describes the whole process of developing such application from analysis and practical examples and descriptions of the implementation of all the laid down goals to some additional theoretical options for future improvements.

Keywords Detection of plagiarism, Django, Python, React, GraphQL, TypeScript, Next.js, JavaScript, Data visualization, Abstract syntax tree

Contents

List of Figures

Introduction

In this current time of Covid-19, many schools chose or were forced to teach their students online, often with no way of any physical contact, not even for assessing the student's performance. With exams, homework and other assignments being undertaken at home, it became more effortless than ever to cheat on a test or copy someone else's homework, and it is the school's responsibility to implement such measures to ensure that the conditions are the same for everyone.

There are many ways to stop cheating but to ensure a level playing field for everyone is to have multiple measures in place during the exam and then to have the necessary tools to evaluate and catch culprits even after the assignment is already submitted.

This work is a part of the LearnShell web application, which is a system for managing, entering and performing programming exams and assignments in scripting languages. The application is made out of multiple modules. The back-end module is built in Python, specifically with the framework Django and offers a GraphQL API. The front-end module is built with TypeScript using React.

The goal of this work is to create a tool to help teachers find plagiarism among the solutions and the culprits responsible for them. It is also essential to help teachers not only find plagiarism but also give them a bigger picture of the situation. A lot of the time, the culprits do not just randomly copy each other, but they often cooperate long term. Such information is crucial in recognizing patterns and then reacting accordingly.

As this work is focused on detecting plagiarism in scripting language assignments such as Shell, which can be very short, or there could also be some very obvious solution, the chances are that there will be some false positives. Helping finding these false positives is also critical.

LearnShell has already implemented a very basic plagiarism detection algorithm that can detect a 100% matching submissions. Even with such a basic solution, this algorithm has caught many students cribbing their solution. The goal of this work is to improve this algorithm to catch students that try to outsmart it.

In the end, the final result of this work should be a fully functional web application with a focus on the user experience to help teachers do this work effectively so that it does not distract them from the more important part of the job, teaching.

Chapter 1

Essential concepts and technologies

This chapter explains and defines the most vital concepts and technologies that are used throughout this whole work. In some parts of this work, it might be critical to understand these concepts at least on some fundamental level to fully grasp the ideas and better understand why a particular solution was chosen.

1.1 Theoretical concepts

This section defines some of the terms and concepts used in this work and also explains their relevancy.

1.1.1 Plagiarism

"Plagiarism is presenting someone else's work or ideas as your own, with or without their consent, by incorporating it into your work without full acknowledgement. All published and unpublished material, whether in manuscript, printed or electronic form, is covered under this definition. Plagiarism may be intentional or reckless, or unintentional. Under the regulations for examinations, intentional or reckless plagiarism is a disciplinary offence"[1], that is the definition from the University of Oxford. The forms of plagiarism can take many forms.

Paraphrasing plagiarism is the first severe type of plagiarism. It means that the author did not really copy the text itself but rather used the original idea as his own. Often times the author will copy the text and then change the structure, order of words or sentences, etc. and then present this as his own work without citing the original author. A more extreme case of paraphrasing plagiarism is **verbatim plagiarism** which means that the plagiarist copied part of the original work word for word.

Important to note that this does not mean that authors cannot use ideas from other resources. It just means that authors need to give credit to others that influenced their work by properly citing their work.

Citing works of other authors often times helps to strengthen the ideas, concepts, principles and arguments by showing other additional resources focused on the same topic. It can assure the reader that the concepts that are being explained are universal. It can also give the reader a reference to other resources that focus on the same concepts to help them understand them better.[1],[2]

Programming plagiarism works very similarly in the sense that cribbing someone else's code is against the rules. StackOverflow's TOS say: *"On Stack Overflow all user content, including code snippets, is posted under a version of the Creative Commons Attribution-ShareAlike license."*[3],[4]. This means that one is free to share and change the content (even commercially) if you follow the terms of this license - giving appropriate credit and if you are changing the code or building upon it, you have to do so under the same license as the original.[5]

What is, however, very important to understand is that plagiarism itself has very little to do with licensing. That means that even if the author has done everything right in terms of the license, he could have still committed plagiarism. Plagiarism is more relevant to the university and the course, which sets the rules of what is considered plagiarism. Generally, it is expected that the student must be the *exclusive* author of the programs he submits, but if the assignment is easy enough, this might be very hard to prove as it is very likely that multiple students will come up with the same solution.

1.1.2 Graph theory

This work uses and discusses concepts of graph theory for visualization but also for algorithms and data structures that can be used to create very sophisticated solutions for detecting plagiarism.

A **graph** G is a finite, nonempty set V of nodes (also called points or vertices) together with a set E of unordered pairs of distinct nodes of V. Each pair $e = \{u, v\}$ from E is a edge (also called an line or a link). Two distinct edges are **adjacent** when they share a common node.[6] One can visualise such graph as a set of nodes connected by edges, an example is in Figure 1.1.

A **path** is a set of distinct adjacent edges leading from point u to point v. A **cycle** is a set of distinct adjacent edges leading from point u to the same point u. There are some important properties that some graphs have. A graph is **connected** if every pair of points is connected by a path. A graph is **acyclic** if it has no cycles; otherwise, it is cyclic. A **tree** is a connected acyclic graph. For example, the graph in the Figure 1.1 is connected because there is a sequence of edges leading from each node to every other node and has one cycle made out of nodes 2, 3 and 4.

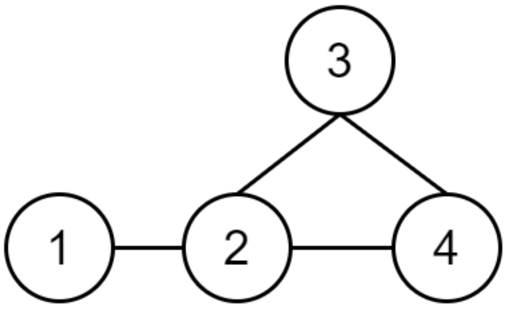


Figure 1.1: An example of a graph with 4 nodes and 4 edges.

One can then choose a single point and declare it as a root of the tree. Trees are usually drawn with the root at the top; a child of a node is such a node that is further from the root than the node itself (then this node is a parent of its children).[6] The following example is a graph that is a tree because it is connected, has no cycles and its root is the node numbered 1.

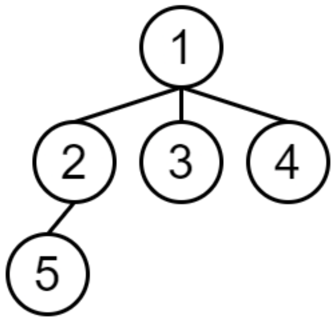


Figure 1.2: An example of a tree with 5 nodes.

1.1.2.1 Abstract syntax tree

An abstract syntax tree (AST) is a tree that represents the structure of a source code written in some programming language. Each node represents a construct used in the language and its children are parameters/operands of the construct. ASTs are free of inessential information about the source code itself - there is usually no punctuation and delimiters (braces, semicolons, etc.) or whitespaces, all of these characters are discarded, but the idea of the program is still preserved in the structure of the AST, for example, the meaning of a semicolon that separated two commands from each other is still preserved in the form of the two separate nodes in the AST.

ASTs are usually the result of syntax analysis. They are used in many different tools, such as compilers, but they are also used for static code analysis, where the tools can go through the AST and find errors or patterns without executing the code itself.[7], [8]

1.1.3 Functional programming

Functional programming is a programming paradigm - a style of programming; it introduces different concepts that can be used to make cleaner and more manageable code. The main ideas of functional programming are to avoid code with shared state, mutable data and side-effects.

Shared state is a variable that is declared in a shared scope or is being passed between scopes. Functional programming avoids this by deriving new data from the existing data.

Mutable object means that the object can be modified even after its creation, but functional programming allows only immutable objects that cannot be modified after they are created.

Side effects are any changes that functions make that can be observed outside of them other than their return value. A very obvious example is pretty much any function that does not return anything. Such function either does nothing meaningful or has side effects. An example of such function could be very simple:

```
function PrintHelloWorld() {
    console.log("Hello World")
}
```

This function's side effect is that it prints the text "Hello World" to the console, which can be observed outside the function itself. It is clear that in some cases, side-effects are unavoidable (e.g., storing data in a database). Functional programming's approach to these kinds of functions is to isolate them and keep them separate from the program's logic.

This is possible because of **higher order functions** - these are such functions that take other functions as an argument, return a function, or both. Such functions make the code very elegant, short and readable, but importantly it makes the code very reusable. An example of such function/method is `map()` or `filter()` which are methods in JavaScript that can be called on an array.

```
[1,2,3].map(x => x * 2) // [2,4,6]
[1,2,3].filter(x => x % 2 == 1) // [1,3]
```

The `map()` method takes a function that describes how each element should be changed, or in other words, what each element should be mapped to. The result of the `map()` method is a new array with the mapped elements.

The `filter` method takes a predicate function (a function that returns a boolean value). The result of this method is a new array with elements that satisfy the predicate function.

In the end, functional programming has some big advantages if done correctly. One of the advantages is apparent when dealing with concurrency and parallel programming. With all of these guarantees that were mentioned, the order of computations in most cases does not matter. It also prevents race conditions and deadlocks. Another great advantage is that it saves time during testing because it is much easier to test code written in a functional way but more on that in the chapter about testing.[9]

1.2 Technologies

This section describes the front-end and the back-end technologies used in LearnShell.

1.2.1 Django

"Django is a high-level Python Web framework that encourages rapid development and clean, pragmatic design. Built by experienced developers, it takes care of much of the hassle of Web development, so you can focus on writing your app without needing to reinvent the wheel. It's free and open source."[10]

Django is not just a framework that helps developers build web applications, but it makes sure that the project has a certain structure that is **maintainable** and expandable in the future.

Django is a very **versatile** framework as it allows developers to build any type of web application that will cooperate with many different setups - it can work with different client-side frameworks, deliver content in any format, communicate with many different databases, etc.

The Django web applications are also **secure** as the framework implements some security measures automatically - one example of this is that Django stores the session information in the database and shares a key in cookies rather than the entire session data, which is a common mistake developers make. Another automatic security measure is storing a password hash in the database instead of the raw password, which is another common security mistake. Django also implements protection against many more vulnerabilities such as SQL injection, cross-site scripting and many more.

Each part of the Django project is independent of the others, making the code much more maintainable but also more scalable. Traffic across the whole application can be tracked to determine which part of the application (caching servers, database servers, application servers) needs more resources and then provide it only to that module.[11]

1.2.2 API Architectures

API (Application Programming Interface) is an interface that allows two software applications to communicate. In the context of web applications, the application that runs in the browser, also called the client, can communicate with the server using the server's API. This way, the client can, for example, request data to display them, send a request to add a comment or delete one. There are several architectonic styles to choose from.[12] The most common one is REST, but in 2015, Facebook released GraphQL, which gained a lot of popularity since then.[13]

1.2.2.1 REST

REST APIs are such APIs that also satisfy the rules of REST architectural style. The most important rules are that requests to such API are made through HTTP, and no client information is stored between requests. Each resource is accessible by a unique identifier (e.g., URL), and there is a defined way to manipulate the resource the way the client wants. The resource that is then sent back as a response can be sent in many different forms such as XML, HTML, Plain text, but the most common is JSON.[14]

An example of such API is one from `hacker-news.com`. Making a GET HTTP request on the following URL: `https://hacker-news.firebaseio.com/v0/item/12345.json` will result in a response from the server that will look like this:

```
// Sending GET request to:
// https://hacker-news.firebaseio.com/v0/item/12345.json
// The reponse:
{
    "by" : "bootload",
    "descendants" : 6,
    "id" : 12345,
    "kids" : [ 12426, 12505 ],
    "score" : 6,
    "time" : 1176412001,
    "title" : "Distributed file storage: MogileFS",
    "type" : "story",
    "url" : "http://www.uncov.com/
            2007/4/4/
            distributed-file-storage-with-mogilefs"
}
```

Figure 1.3: An example of a response from HackerNews REST API.

The result is an item with the idea of `12345` in the JSON format as specified in the URL. It is possible to change the operation the server should make. By making a different type of HTTP request, such as a DELETE request, one can indicate what kind of action the server should make. Now, as this is a public API, this is obviously not supported, but the general idea is to distinguish the separate CRUD operations by their respective HTTP requests:

- Create - POST
- Read - GET
- Update - PUT
- Delete - DELETE

An important thing to note, especially in the context of GraphQL, is that the response from a REST API is fully determined by the type of the request and the URL to which the request was sent to. This has big limitations as no matter what the client's intent is, the data is still sent the same way. Even if the client only wants to display the title of the post, it is not possible to fetch only the title; the server will always send the entire resource unless there is an URL that specifically supports fetching only a specific information.[15]

1.2.2.2 GraphQL

"GraphQL is a query language for APIs and a runtime for fulfilling those queries with your existing data. GraphQL provides a complete and understandable description of the data in your API, gives clients the power to ask for exactly what they need and nothing more, makes it easier to evolve APIs over time, and enables powerful developer tools."[16]

GraphQL is essentially a new way of thinking about APIs. Its main idea is to solve the problem of over-fetching or under-fetching by introducing a query language and a server-side runtime for executing these queries. The API is built by defining the data - what data should the API offer and what fields and type fields the data have. To fetch the data, the client has to describe what fields are needed, and GraphQL will send the response in that exact structure[17]; an example of such communication can be found in the Figure 1.4 and 1.5.

Depending on how much information the server stores about each of the countries, this can potentially save a lot of bandwidth and also save space for the client.

```
// Sending query to: https://countries.trevorblades.com/
query {
    countries {
        name
    }
}
```

Figure 1.4: An example of a GraphQL query.

```
// The response:
{
    "data": {
        "countries": [
            { "name": "Andorra"},
            { "name": "United Arab Emirates"},
            ...
        ]
    }
}
```

Figure 1.5: An example of part of the response the server gives after the previous request.

1.2.2.3 GraphQL vs REST

The biggest advantages of GraphQL are very obvious as it was designed to solve the issues of REST APIs - querying the data by exactly describing the structure of the data the client wants solves both the **over-fetching** which happens when the client only needs some of the information, but the server sends more because it does not know what exactly the client wants. It also solves **under-fetching** which happens when the client needs more data than it received. This happens especially when fetching a list of items, and the client needs to send additional requests to fetch all of the items it wants.[18]

Using GraphQL leads to much faster front-end development because it is very **flexible**, but with REST, there might be changes on the front-end that are not yet reflected on the back-end, and so the back-end does not support sending the required information.[18]

Another advantage is **versioning** - GraphQL's APIs usually do not have to be versioned because the API only returns the data that is explicitly requested, and new capabilities can be added through new types and fields without breaking the previous solution. REST APIs, on the other hand, have to be versioned because with every change to the API, the previous end-points might break. Even the `hacker-news.com` API (from the example in Figure 1.3) is ready for this as it already have the version of the API in the URL (‘`.../v0/...`’).[19]

The main disadvantage when it comes to GraphQL is **performance** on the server’s side. There is some overhead when it comes to parsing, validating and then executing the query. The server has to parse the query into an AST then validate it against the schema and then execute the AST starting from the root - collecting all the data and returning the requested JSON. All of this takes additional time, but on the other hand, the amount of requests is lower with less data that needs to be transferred.[20]

1.2.3 TypeScript

TypeScript is an open-source language which is built on one of the most popular programming language - JavaScript. As the name suggests, TypeScript adds types to JavaScript, which provides a way to describe objects and variables to help the productivity of the developers as they get better code validation.

There are no real downsides to using TypeScript because it is not needed to write everything in TypeScript or rewrite already written code to TypeScript as a valid JavaScript code is also a valid TypeScript code and will work the same way. But obviously, by not using the features of TypeScript, one does not gain any benefits. This also means that a project that does not currently run on TypeScript can be slowly converted to a project that is unrecognizable from one which was built on TypeScript from the beginning.[21]

One of the biggest advantages of TypeScript, as hinted above, is error reporting that is provided by the compiler that has much more options to check for errors with TypeScript code compared to JavaScript code. This leads to errors being discovered earlier.

It also makes the code much more readable because static typing gives the code more structure. The IDE also has much more information and so it can give smarter hints, autocompletion, etc.

As already said, there are no real disadvantages to using TypeScript, but there are things to be aware of. TypeScript’s types are only checked at compile-time, and after the code is transformed into a regular JavaScript code, the types get removed by this process.

Even though TypeScript helps to prevent some errors, it is not a replacement for testing. Comprehensive testing is still needed as in any other language, even if the code is less prone to bugs. Also, sometimes the code might get a little messier by introducing types as there is more code to maintain and also working with types, classes and interfaces might lead to over-engineered code. Especially new team members and beginners might get lost, especially if they have no previous experience with other statically-typed languages.[22]

1.2.4 React

React is a declarative, efficient and flexible JavaScript library for building web applications and user interfaces. It was released by Facebook in 2013, to address the challenges of building large-scale, data-driven web applications. Its primary goal is to minimize bugs that developers make when building user interfaces. Possibly the main idea that React uses to achieve this goal is using components. Components are self-contained, logical pieces of code that describe one portion of the user interface. Furthermore, they are then used to assemble the whole UI of the application. This makes the code very reusable as one component can be used in many different parts of the application. Each component can receive data from its parent component through props to make the component even more customizable and versatile.

Components make the code more maintainable as each part of the application is isolated, which also makes the development faster as the developer can purely focus on just the one component and its functions and features instead of the application as a whole.

React utilizes JSX which is an extension of the JavaScript language so that JavaScript can work with HTML more directly[23], for example:

```
const heading = <h1>Hello  World!</h1>
```

React will then take this JSX and compile it using Babel into a JavaScript code, like this:

```
const heading = React.createElement(
                "h1", null, "Hello World!")
```

React is one of the most popular technologies for making web applications as it is quite easy to learn and also easy to use, even more so now when there are countless resources, books, articles and tutorials. But the React team did create excellent documentation with examples and tutorials.[24], [25], [26]

1.2.5 Next.js

"Next.js gives you the best developer experience with all the features you need for production: hybrid static & server rendering, TypeScript support, smart bundling, route pre-fetching, and more. No config needed." [27]

Next.js is a React framework that solves many common problems React applications have in common, and it does so **without any configuration** that needs to be set up by introducing a certain structure which also makes the software more maintainable and clean in the future.

A big feature of Next.js is **automatic routing** - any files in the `pages` folder will be automatically mapped to its URL based on their path in the filesystem. This feature saves a lot of time otherwise spent on setting up the routes. Another time-saving feature is **hot code reloading** - when Next.js detects any changes in the files, it will reload the page in the browser automatically, which saves a lot of time, especially when the changes are frequent and small.

Next.js also helps with performance in a few different ways. First of all, it allows **server-side rendering** which means React components can be rendered server-side and then sent as HTML to the client. Another feature is **automatic code splitting** which makes sure that only the code that is needed for the particular page the user is requesting is sent to him, instead of sending the whole application's code at once. On the other hand, it supports **prefetching** to automatically prefetch resources (including the code missing due to code splitting) to other pages.

Ecosystem compatibility is also a big advantage to this framework as it plays well with React by fixing some of the common problems React developers have to face. The whole framework is written in TypeScript, which gives it excellent **TypeScript support**. [28]

Chapter 2

Analysis

In order to build software that is going to be meaningful, intuitive and useful, one needs to analyze the environment and the options available. This chapter focuses on analyzing the previous solution, the context of the whole LearnShell application and its modules and the available options that could be used and why.

2.1 Existing solution

LearnShell is a web application that was built for helping to teach students scripting languages. The main focus (especially for now) is on helping students understand how to write shell scripts. The following schema describes the architecture of the application:

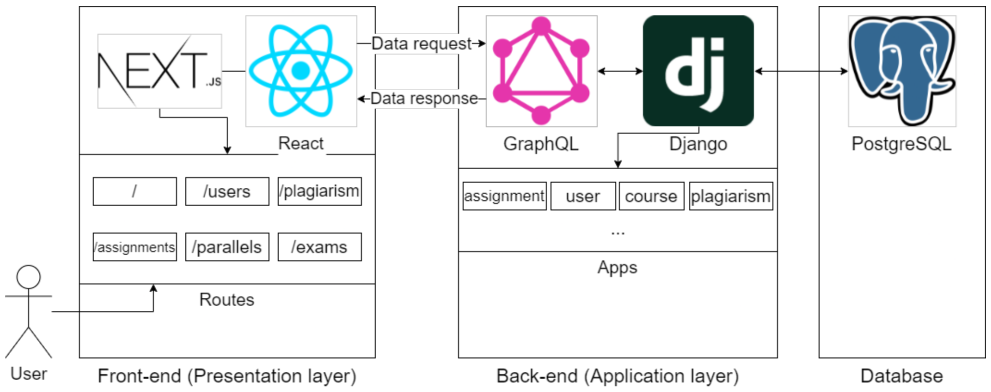


Figure 2.1: Interface schema of the LearnShell's architecture.

The whole application is split into a couple of modules. The most important module is the back-end built on Python and its framework called Django, which communicates with other modules but also with the database.

The other important module is the front-end web application which is built on React and also utilizing the features of some modern frameworks such as TypeScript. The HTTP endpoints are exposed using the Next.js framework, which handles all the routing.

LearnShell has already got a working detection algorithm that can take two scripts and compare them to each other, and if and only if there is a 100% match, the algorithm will evaluate it as plagiarism. If only one character is different, even if such character has no effect on the function of the script, it is not going to get detected by this algorithm.

The results of the detection process are then stored in a JSON file in the following structure:

```
{
    "script": string,
    "culprit_assignment_name": string,
    "culprit_count": number,
    "culprits": Array<string>
}
```

The first field named script contains the solution that was detected as a duplicate solution because at least two students submitted such solution. Then there is the culprit_assignment_name field, which contains the name of the assignment that the students were solving. The most important field is the field named culprits which is the actual list of culprits that had the same solution and the culprit_count field is the length of this list.

This JSON file is also the only way to access the results of the detection algorithm as LearnShell does not offer any front-end application to view these results in any other way. This makes it much harder for teachers to find their way around the data and what it is actually saying. The teachers might want to make some other analysis to understand the data, look for relationships between the students, find patterns and also identify if the suspicion in the data is a true positive or a false positive.

2.2 Requirements

This section contains all the functional and non-functional requirements for the solution that this work should be able to satisfy.

2.2.1 F1 - Improving the detection algorithm

The detection algorithm should be able to detect cribbed submissions from students even if they are not a 100% match.

2.2.2 F2 - Display duplicate submissions

The plagiarism data should be able to be displayed on the front-end of the application.

2.2.3 F3 - Display duplicate submissions of a particular student

The front-end should be able to aggregate the data and show only the plagiarism data of a chosen student.

2.2.4 F4 - Display duplicate submissions of a particular assignment

The front-end should be able to aggregate the data and show only the plagiarism data of a chosen assignment.

2.2.5 F5 - Resolving suspicions

The teacher should be able to resolve these suspicions.

2.2.6 F6 - Exporting the data

The data should be exportable into a CSV or a JSON file.

2.2.7 F7 - Data visualization

The front-end should be able to visualize the data to show the relations between the culprits.

2.2.8 NF1 - Front-end extends on the existing solution

The front-end should follow up on the already existing solution built on React and Next.js.

2.2.9 NF2 - Command-line detecting tool

The detection tool should be built as a Django Command and be part of the existing Django application.

2.2.10 NF3 - Responsivity

Although responsivity is not the main focus due to the lack of use cases in which this feature is needed or even noticed, it is still a feature that is expected in any modern web application. Also, it is important to note that responsivity does not only mean desktop/mobile size support. It means that the web application can adapt to any screen size. In the context of this work, there might be some use cases in which the application is not usable due to its lack of responsivity.

The resolution of the screens in the classrooms might differ from one to another. While the device used is still a desktop computer, the screen resolution might drastically change from an old HD monitor to a Full HD monitor or even to a 4K monitor. Teachers might also change the size of the browser, especially when comparing multiple scripts, users or assignments.

All of this leads to the necessity to make the whole solution fully responsive.

2.3 Proposed solution

Although the current detection algorithm works and has found many students that had the same solution, it is obvious that it does not take much effort to outsmart it. Simply adding one more whitespace is enough to go undetected by this algorithm. On the other hand, if the algorithm does detect a cribbed solution, it is very likely that it actually is a true positive because the solution is just simply the same.

There are several ways to improve this algorithm with some advanced data structures such as AST. Unfortunately, the tools available in this field are very limited, especially in the context of Shell, and as this work is not only focused on the detection algorithm, there will be no resources for us to implement a fully working solution utilizing ASTs. For this reason, the whole detection system must be very flexible and, most importantly, expandable for it to be easily improved and built upon in the future.

The algorithm itself will not always be able to decide if the two scripts it is comparing are really cribbed, especially when only a part of one script is identical to a part of the second script. For this reason, the detection algorithm should return the likelihood of these two scripts being cribbed rather than returning a yes/no answer.

The main improvement that has to be done on the front-end is to add the option for teachers to access these detection results and display them in such a way that helps them decide better if the suspicions are true positives or not.

Having just the pure data displayed is not enough in this case, as the patterns formed in the data will not be clearly visible. The solution must be able to sort the data, filter and aggregate it in such a way that the teacher will be able to find the patterns and be able to resolve these plagiarism suspicions more effectively.

To see some additional patterns, we propose to display the data in a more visual way than in just pure text to able to see patterns that might not be obvious otherwise.

2.4 Choosing a data visualization library

Data visualization is a key part of helping humans understand the data they are working with. Modern technologies allow us to process a lot of data and visualize them by using images, maps, charts, etc. One of the many vital parts for data visualization to be useful is to choose the right type of visualization. Choosing the wrong type of chart or configuring the chart poorly can lead to the visualization being useless in the better case and being misleading in the worse case.

We tried to find a solution for teachers to better understand the plagiarism data that LearnShell detects. For teachers to do an effective job in resolving suspicions, they need to know the relations between the culprits. Our goal with this visualization is trying to show the teachers these relations and how strong these relations are with respect to each other. Therefore we decided that the best visualization tool is a network graph.

This structure can be used to represent a lot of types of data. We can use it to visualize the relations (represented as edges) between the culprits (represented as nodes), hoping to show not just the relations themselves but also other types of information - changing the color, thickness and the border can also display some additional information.

Out of all the possible libraries for displaying network graphs, surprisingly, there are not that many of them that support changing the thickness of the edges, which was one of our main requirements. Our other requirement was for the library to be easily implemented into React. In the end, we were choosing between D3.js, vis.js and nivo.

2.4.1 D3.js

“D3.js is a JavaScript library for manipulating documents based on data. D3 helps you bring data to life using HTML, SVG, and CSS. D3’s emphasis on web standards gives you the full capabilities of modern browsers without tying yourself to a proprietary framework, combining powerful visualization components and a data-driven approach to DOM manipulation.”[29]

D3.js is a foundation for a lot of other libraries that are built upon it. In most situations, it is better to use these libraries as they offer all the basic features one would need; however, D3.js has its place when there is a need for a very specific feature that other libraries do not offer on their own because

D3.js is very flexible.

Pros:

- Flexibility
- Examples - this library is very widely used which makes finding examples very easy

Cons:

- Steep learning curve
- Documentation is not clear
- Not initially intended for React

2.4.2 vis.js

"A dynamic, browser based visualization library. The library is designed to be easy to use, to handle large amounts of dynamic data, and to enable manipulation of and interaction with the data."[30]

Vis.js is much easier to use than D3.js while preserving a lot of the flexibility as D3.js. While the design is not great out of the box, it can be customized to look better. The biggest disadvantage of this library is the vis-react library which should make it easier to connect up with React, but it is very poorly documented, which makes it really hard to use.

Pros:

- Still remaining flexible
- Customizable
- Easier to use

Cons:

- Not initially intended for React
- Hard to use with React
- Documentation is very poor (espeacially on the React library)

2.4.3 nivo

"Nivo provides a rich set of dataviz components, built on top of the awesome d3 and Reactjs libraries."[31]

We believe this to be the best option that we decided to use in the final implementation as it offers the best of both worlds and is specifically built to use with React. Also, the documentation is well made with some interactive examples, which makes implementing a solution that does what is needed very simple to do.

Pros:

- Offers all the needed features
- Is built for React
- Documentation
- Easy to use

Cons:

- Not as flexible

Chapter 3

Implementation

This chapter focuses on the implementation part of this work - what was implemented and the pitfalls encountered during this process.

When implementing all the features, creating the components and designing the codebase, the idea that we kept in the back of our minds was to concentrate on building a solid foundation for future expansions and improvements.

LearnShell uses GitLab for its Git repositories for version control, and the repositories can be found here:

- Back-end: `https://gitlab.fit.cvut.cz/learnshell-2.0/ls`
- Front-end: `https://gitlab.fit.cvut.cz/learnshell-2.0/ls-web`

3.1 Detecting plagiarism

Keeping in mind the future improvements that can be done in this part of the application, we decided to build a firm structure that could be easily expandable in the future by adding more advanced algorithms for the detection of plagiarism.

We decided to split the plagiarism detection into multiple steps. The first step is to preprocess the input using algorithms to transform, reduce or process the input in some way. There can be a list of these functions provided to do multiple changes to the input. If no such functions are provided, then this step is skipped, and the two scripts are used without any changes made.

The second step is to take the preprocessed input and run the detection algorithm that compares the two inputs and returns an output that is a number describing the likelihood the two inputs are cribbed - 0 being the least likely and 100 being the most likely.

The third step is to process this score by an additional function. This could be used for an easing function to make lower scores even lower and higher scores higher. This score is then used to calculate the final score as a percentage of the maximum score this scoring process can give.

$$Final\ Score = \frac{Score \cdot Max\ Score}{100}$$

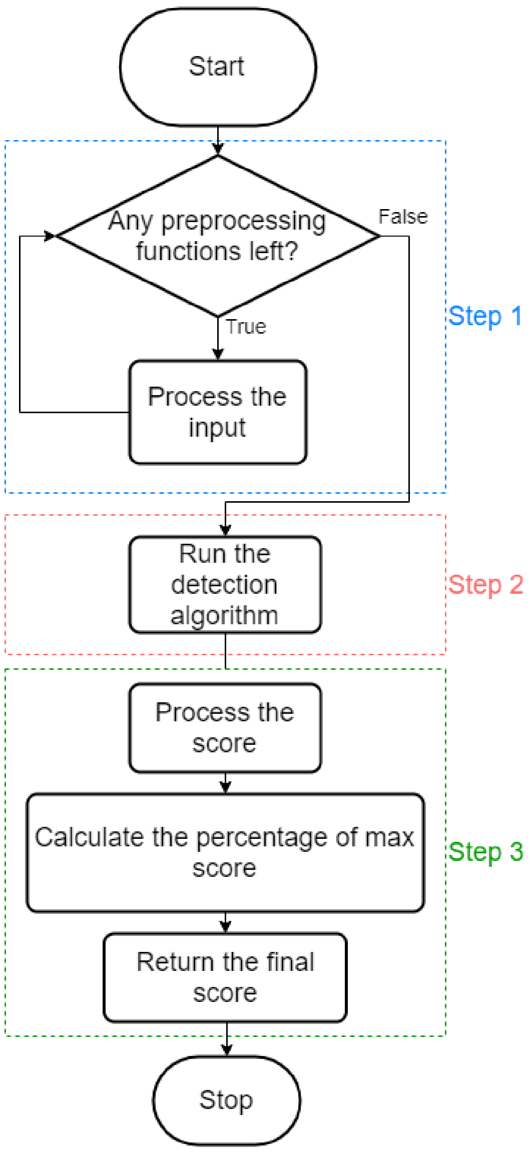

Start
Any preprocessing functions left?
False
True
Process the input
Step 1
Run the detection algorithm
Step 2
Process the score
Calculate the percentage of max score
Return the final score
Step 3
Stop

Figure 3.1: Flowchart of the process of detecting plagiarism.

All of this is done by a decorator function that takes the `max_score`, `score_decorator` and the list of `input_modifiers` as a parameter and uses them in a wrapper function.

```python
def decorate_score(max_score, score_decorator, *script_modifiers):
    def detection_decorator(detection_function):
        def wrapper(script1, script2):
            for script_modifier in script_modifiers:
                script1 = script_modifier(script1)
                script2 = script_modifier(script2)
            score = detection_function(script1, script2)
            score = score_decorator(score)
            return score * max_score / 100
        return wrapper
    return detection_decorator
```

The code above is an example of such implementation. After providing the necessary parameters to the `decorate_score` function it returns a function that takes the `detection_function` which is then called with the 2 input scripts.

To add even more flexibility to this solution, we decided to add multi-scoring functionality - multiple detecting (scoring) functions will analyze the inputs and return the score in the same way as described before. The whole process will then output an array of scores that can give much more insight. To evaluate two scripts as cribbed, one can then simply take the maximum from the array and see if it exceeded a certain threshold. There is, however, much more that can potentially be done with the array, as evaluating the scores in a different way might be very beneficial.

This resulted in a class `PlagiarismDetection` that contains this array of scoring functions. This class supports adding scoring functions by providing all the necessary parameters. After everything is ready, the `run_detection` method can be called to run all the scoring functions returning the array of all the scores from the scoring functions.

It is also possible to run the detection functions one by one by specifying the index of the function that should be called. This can enable a much more sophisticated way of running the detection. This solution can also result in faster evaluation as the detection can be stopped if it is clear that the two scripts are cribbed and save time by not running more advanced detection algorithms.

3.1.1 Script modifiers

Two modifiers have been implemented as a result of this work. The first one is a function that removes all the spaces, newlines, tabs, etc., from the original script and returns the script with them removed. It is important to note that it removes all of them, not just the duplicate whitespaces (reducing 2 or more consecutive whitespaces into only 1). This is intentional because of the following example. If a student were to crib another's student assignment and tried to cover this by adding non-meaningful spaces to make the script unique enough without changing the functionality, and this modifier did not remove all the whitespaces (and instead just used the reducing option), this attempt could be successful. Let's imagine the original script looks like this:

```
ls|sort␣-r|head␣-2
```

An attempt that would still be undetected by a standard string equality algorithm would be to add spaces around the pipes:

```
ls␣␣|␣␣sort␣-r␣␣|␣␣head␣-2
```

These spaces would not get removed but only reduced to one space, and as a result, the string forms of these two strings would not match.

The second function is a function that converts all the upper-cased letters in the script to their lower-case equivalent; the rest of the characters remain the same. This makes the whole future analysis **case-insensitive**.

3.1.2 Score decorators

We implemented 2 easing in functions and 3 easing in and out functions. These functions could be used to make lower scores matter less, which might be useful, especially when dealing with Shell scripts that can be quite similar just by the nature of the structure of these scripts.

For example, when the assignment says to get the content of a file, manipulate it in some way and then sort the result, it is almost inevitable that all the students will write very similar code to get the content of the file and then to sort the result.

To reduce the final score of such solutions with these similarities, we can use easing functions to diminish the impact of such small similarities. This solution might be an elegant way of later avoiding going through lots of false-positive suspicions that were caused by the fact that the approach to solving at least part of the assignment was too obvious.

The implemented easing functions are:

- Ease in quadratic
- Ease in cubic
- Ease in and out quadratic
- Ease in and out cubic
- Ease in and out exponential

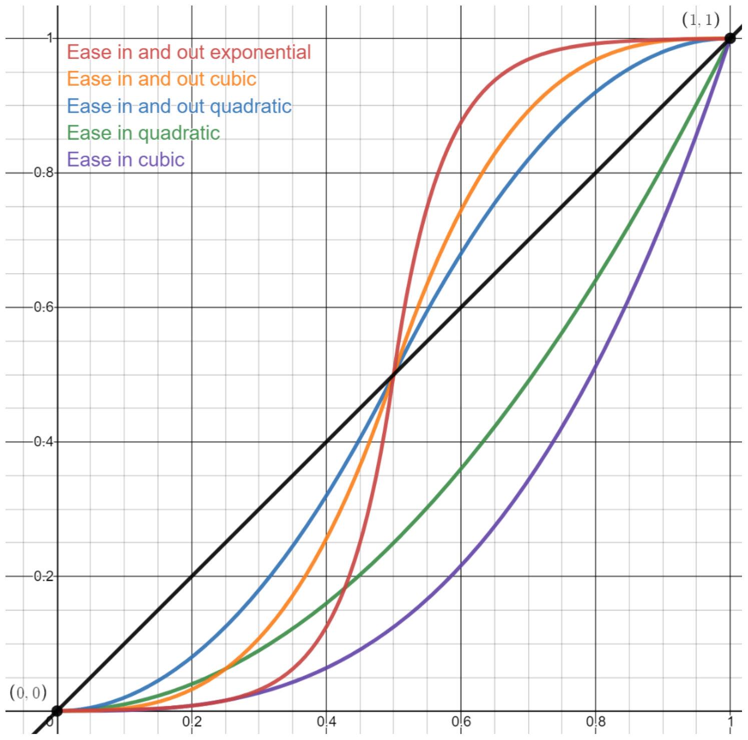


Figure 3.2: Graph of the implemented easing functions.

3.1.3 Detection algorithms

The most basic algorithm that we implemented is a simple string equality algorithm that returns a score of 100 when the scripts are identical, and 0 if one or more characters are different. This solution is suddenly much more powerful thanks to the previous step that the script has to go through. If the previous preprocessing removed all the whitespaces and converted the script to lower-case, the string equality algorithm suddenly becomes a case-insensitive solution ignoring all whitespaces.

To make the algorithm even better, we decided to use the longest common substring (LCS) algorithm. With this solution, we can find the biggest part of the two scripts that they have in common and translate this length into the score. This is done by calculating the ratio between the length of the LCS and the length of the shorter script. This ratio is then scaled to be in the correct range `<0; 100>`.

$$Score = \frac{LCS\ length}{min\{Script\ 1\ length,\ Script\ 2\ length\}} \cdot 100$$

As a result, this algorithm is an extension of the string equality algorithm because if the two scripts are entirely the same, then the result of the LCS will be the full script, and the ratio will be equal to 1 as both the lengths are the same, resulting in the score to be 100.

After testing the function, we made a second version to be more fitting for the context of Shell scripts. The main flaw of the algorithm was apparent, especially when dealing with shorter scripts (under 30 characters), as these scripts were scoring quite high because of how common some constructs in these scripts are.

A very simple example of this was when the students were asked to read the contents of a file to then process. Pretty much every solution of every student will start with a very similar or identical construct: `cat <file> |` and if the scripts are short enough, such substring will make up a big percentage of the final script.

To counteract this flaw, we decided to introduce a minimal length of the substring that will nullify the results of the algorithm if the length is under the threshold. The threshold is determined by the length of the scripts.

3.1.4 Future improvements

This work is supposed to be a solid foundation for any future improvements to make the detecting algorithm even better. A really big step forward would be to stop treating the scripts as strings. As mentioned several times in this work, Shell scripts but any programming code has some constructs and structure. Trying to find, extract and work with this structure would greatly improve the possibilities for the detection algorithm.

Such structure is obtainable by doing a syntax analysis, and as a result, we receive a syntactic tree. The advantage of turning scripts into syntactic trees is that they perfectly describe the scripts without any redundant information. But to fully tap into the potential of this solution, it is needed to use an abstract syntax tree (AST) instead. The main difference is that ASTs describe the meaning of the code independently of a particular syntax.

To compare these structures, we can use hashing by calculating a hash for each AST node taking into consideration not only the node itself but also its children. Such hash then describes not only the node itself but an entire subtree. When we are then trying to find similarities in the code, we can compare the hashes to find parts of the tree that are similar.

There are many possible improvements to this solution as well. One of which might be to detect parts of the scripts that are common across many solutions and ignore them when comparing the ASTs. Other improvements could be made when creating the AST. The AST can be treated as an ordinary tree, allowing us to use common graph algorithms. We can use tree shaking or algorithms to minify or optimize the tree and compare these versions instead.

We believe a solution of this sort would be great to implement in the future because it would be immune to any modifications such as renaming variables, condition/statement reordering, adding additional characters (whitespaces), etc.[32]

3.2 Displaying plagiats

The apparent improvement on the front-end side of things is to build a better user interface for displaying the data. The only way in the previous solution was to look at the data directly in the JSON file. To make it more user-friendly, we decided to display all the data in the front-end of the application itself but improve the readability and accessibility.

To create a clear separation between each part of the front-end application, every logical piece of the application will be in its own component. This drastically improves the reusability of the code that is written because each component can be placed in several parts of the application without writing more code.

The first component is the list of suspicions that will contain all the suspicions that the detection algorithm found and display all of the information about each one. To display all of the suspicions, we used the `map()` function that has already been discussed in the chapter about functional programming. Every item from the list is going to get mapped to JSX, which React can then display. The basic implementation can look like this:

```
suspicions.map(suspicion => {
    <h1>{suspicion.culprit_assignment_name}</h1>
    <code>{suspicion.script}</code>
})
```

The JSX is very similar to HTML but has some additional features. One of the examples of this is `{ data }` - this will actually print the contents of the variable `data` instead of what is literally written in the JSX. To display the culprits, the same technique can be applied as the data about the culprits is also an array.

```
suspicions.map(suspicion => {
    <h1>{suspicion.culprit_assignment_name}</h1>
    <code>{suspicion.script}</code>
    <div className="culprits">
        suspicion.culprits.map(culprit => {
            <span>{culprit}</span>)
        }
    </div>
})
```

Important to note, in JSX `class` is a keyword and cannot be used the same way as in HTML. The equivalent in JSX to `class` is `className`.

Outputting extensive lists of data like this could be chaotic; the list can also overflow horizontally or be very long if there is only one item on one line. To fix any issues like this, we found using flexbox was very useful in this regard. Flexbox is a module of CSS3 that helps to align and distribute items in space. There are many different properties one can be set but in this case, setting it up is very simple – enable flexbox on the parent of the list, set the direction to be row and enable wrapping when there is not enough horizontal space:

```
div.culprits {
  display: flex;
  flex-direction: row;
  flex-wrap: wrap;
}
```

There are also two ways to aggregate this data - by the student and by the assignment. This process is straightforward to do because of components. Components can take props where the data can be passed so the component can be reused in different parts of the application with different data.

We can filter only the data containing the selected culprit or the suspicions detected in relation to the selected assignment. After sending this filtered data to the component, the component will just display it as usual.

To distinguish these types of views on the data, we decided to create 3 pages in Next.js. Next.js will determine the URLs that will be used to access this page. It determines it by the file's path. We decided on this structure:

```
plagiarism
├── index.tsx .................................... URL: /plagiarism/
├── assignment
│   └── [AssignmentPage].tsx ..... URL: /plagiarism/assignment/[name]
└── user
    └── [UserPage].tsx ................. URL: /plagiarism/user/[email]
```

On the left side is the path within the project and on the right is the routing that will be used by Next.js. Both the `AssignmentPage` and the `UserPage` have one parameter (indicated in the URL by the square brackets). We can then take this parameter and use it to filter the data. Or in the other direction, we can create a link to a specific user's page by concatenating their email at the end of the URL.

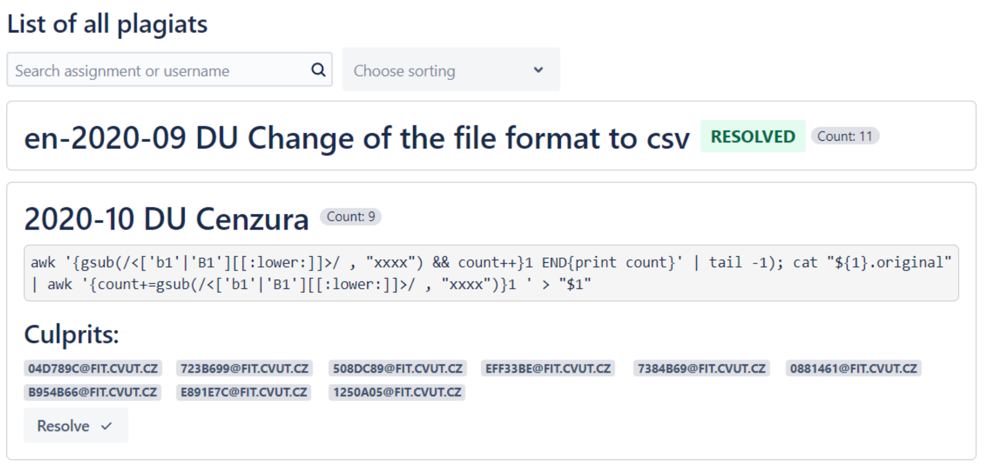
Apart from the suspicions, another prop that is being passed is information about what culprit should be highlighted. This is utilized on the page with data being aggregated around a certain student. We can then highlight the culprit to be more distinguishable among the other culprits.

3.2.1 Resolving suspicions

At the bottom of every suspicion is a button to resolve that suspicion. The button has a `onClick` event that will trigger the resolving process. This process is isolated from the button itself to accommodate for more sources to resolve the data. The isolation will help in the future to change how the resolving is done by changing the resolving process in one place only without breaking the functionality or creating inconsistencies.

The resolving process will find the suspicion in the dataset and add a tag to it `resolve: true` to indicate that the suspicion has already been resolved.

We can then skip suspicions with this tag when displaying the data or rendering them in a different way to distinguish them from the unresolved cases. We decided to design a different solution for these suspicions to make the application more versatile and ready for this situation. This functionality can be easily changed not just by changing how these suspicions are rendered but also by changing how the data is fetched. If we exclude the data with this tag or filter them out at any stage before the render, we can achieve the same end result but with more versatility as the front-end is still ready and capable of rendering such suspicions.



List of all plagiats

en-2020-09 DU Change of the file format to csv

2020-10 DU Cenzura

```
awk '{gsub(/<['b1'|'B1'][[:lower:]]>/ , "xxxx") && count++}1 END{print count}' | tail -1); cat "${1}.original"
| awk '{count+=gsub(/<['b1'|'B1'][[:lower:]]>/ , "xxxx")}1 ' > "$1"
```

Culprits:

04D789C@FIT.CVUT.CZ 723B699@FIT.CVUT.CZ 508DC89@FIT.CVUT.CZ EFF33BE@FIT.CVUT.CZ 7384B69@FIT.CVUT.CZ 0881461@FIT.CVUT.CZ B954B66@FIT.CVUT.CZ E891E7C@FIT.CVUT.CZ 1250A05@FIT.CVUT.CZ

Figure 3.3: The end result of the design of the suspicions.

3.2.2 Table of culprits

Table of culprits is another component that shows the teachers all the culprits in the data and how many cribbed solution they share. This component makes it much easier to recognize what students the teachers should focus on first. It also serves as an index of the culprits, so teachers have an easier time accessing the data aggregated around this particular student.

To improve user experience, the table is sorted by the number of cribbed solutions the student has made in descending order, but the ordering can be reversed or even changed to sort by the username of the culprit. The other user experience feature is pagination. Pagination is necessary because the number of culprits can be over 100, and the table would be too long. This is solved by pagination because there are only 10 culprits being displayed at a time, and the teacher can access other pages using the arrows forward and backward or by clicking the number of the page.

3.2.3 Most common coworkers

When the teacher aggregates the data by an individual culprit, the application offers a table of the most common coworkers in relation to the main culprit. This component helps teachers establish patterns in the data. The table offers them a number of plagiarism the main culprit shares with the coworker as well as how many percent of the main culprit's cribbed solutions have they cooperated on.

The table also offers different ordering and pagination the same way the table of culprits does. Clicking on the username of the coworker leads to a new aggregation based on the clicked user to furthermore examine the relations of the culprits.

3.2.4 Filtering

Filtering is a great way to further enhance the user experience. The amount of data can be overwhelming, and trying to find a particular student or clicking through pages of suspicions could be very time-consuming and frustrating, every component that displays some data has a way of filtering them or ordering them. As every component can access the data through props it can also filter the data separately to each other. The solution has to be effective as there could be quite a lot of data to go through.

For example, to filter culprits based on their username, one can use the ideas from the chapter about functional programming and come up with an elegant solution like this:

```
culprits.filter(culprit => {
    return culprit.toLowerCase().includes(inputValue.toLowerCase())
})
```

Taking the searched term and storing it in `inputValue` variable and then using the filter function to filter out the culprits whose username contains the searched term. Both the username and the searched term are converted to lower-case, which makes the searching case-insensitive. The question is when to call this function as there are several approaches to this.

The first one is to call this function every time the user changes the value in the search box. This is a user-friendly solution that saves the user time because everything is automatic. The big problem with this solution is that the filtering function will be called very frequently, even when the user does not want to, which can also lead to performance issues. The majority of the time, the filtering function will be called unnecessarily, and it can also lead to the browser freezing for some brief moments causing the typing being unresponsive.

The second approach is the obvious fix to this problem – adding a button to confirm that typing the searching term has been completed and the filtering can begin. The problem with this solution is the decrease of the user experience because there is suddenly an extra step that feels unnecessary and can get repetitive and even annoying. Also, adding an extra button might lead to compromises on the designing part as the design might lose structure or be less responsive, especially when dealing with components that can have limited space.

The third approach is improving the first solution by utilizing debounce. Debounce is a function that takes another function `func` and a number `t`. The debounce function calls `func` after the `t` milliseconds have passed since the last call of the debounce function.[33] In practice, this means that the user can type the search term without the debounce function triggering to call the filtering function. After the user is done typing the search term and the chosen time elapses, the filtering will be called, leaving the user satisfied as he did not have to press any buttons and the application still did what it is supposed to.

It is important to choose the debounce time carefully - if too low, the debounce function will not have good enough of an effect, and it will be closer to the first approach that was described. If the time is too high, the user will have to wait and might get impatient or irritated by how slow the filtering is or even might get confused on how to submit the search term as nothing is happening.

3.3 Data visualization

It is very likely that there will be some obvious patterns formed in the data and the relations between the plagiarists. It is essential to find the root cause for why and how the plagiarists cribbed their solutions. There are many reasons why these patterns should form, for example, the students might cooperate because they are friends or have some connections, these groups of friends might be the sizes of 2 or 3 people but can also get above the size of 10 or even bigger depending on the size of the course. Finding these patterns between 2 or 3 people might not be that hard, but with this number getting larger, it becomes much more difficult.

Data visualization helps to combat this issue because — if done right — these patterns should become very apparent. The chosen solution is to display a network graph of all the culprits and their connections. There are two types of objects that the graph is working with - nodes being the culprits and links being the relations between them (having the same solution on at least one assignment). In TypeScript, these two types of objects could be described like this:

```
type Node {
    username: string
}
type Link {
    source: Node,
    target: Node
}
```

We can then iterate over all the suspicions, take all the culprits and find all the combinations of the culprits. It is important because of performance to truly find only the combinations and not permutations because if there is a link from node `u` to node `v` a link from node `v` to node `u` does not provide any additional information and might cause performance issues because every link will get doubled.

To avoid this problem, we created a solution that checks if a link is already included in the graph. If the link is already in the set, then its count gets increased. This number is later used to calculate the thickness of the link.

To fully tap into the potential of using nivo's network graph component, additional information can be added to the graph by changing the appearance of the nodes and links. Nivo's network graph support changing:

- Node's color
- Node's radius
- Node's border width
- Node's border color
- Link's thickness
- Link's color

This can be used to display 6 different additional categories of information. The two most important types of information that should be indicated in the graph is the number of cribbed solutions a student has made, which we decided to indicate by changing the node's radius. The other important information is the number of common cribbed solutions two students have, which is indicated by the thickness of the link between them. We believe this is a very natural decision and that the information is going to get understood at first glance.

The graph is displayed on the page with all the plagiarism as well as on the culprit's plagiarism page (page with the data aggregated by a particular student), where this main culprit will be highlighted by a different color. In this mode, the user can set a depth in which the algorithm will look for other coworkers transitively - e.g., with depth set to 1, the graph will only show the direct coworkers of the main culprit, but with depth set to 2, the graph will also show the coworkers of the coworkers already in the graph.

This is accomplished by an algorithm that finds all the coworkers of the main culprit. The algorithm then does the same thing for all the new culprits found in the previous depth until it is at the wanted depth.

Tooltip is another place where even more information can be displayed. A big drawback of using a tooltip is that it is not visible until the user hovers over a node; therefore, the user cannot access the information stored in the tooltip just by looking at the graph. This is why we decided that this is the place to put the information that would otherwise clutter the graph if displayed directly there – the username of the culprit that is represented by this node as well as the number of cribbed solutions.

3.3.1 Graph settings

Graph settings is a component that makes the graph much more flexible. First of all, the user can change the depth as discussed before but also disable both the dynamic thickness of the links based on the number of common cribbed solutions as well as the dynamic radius of the nodes. Another possibility is to enable the animations to see more clearly how the graph is transforming or disable them to improve performance.

The user can also manually override the repulsivity between the nodes – how much should the nodes repel themselves. Sometimes the graph can overflow, and this is a safety measure, so the user has full control to fix this issue if it happens.

To make this component maintainable and also expandable in the future, the configuration is stored in an object like this:

```
type Settings {
    repulsivity: {
        func: (nodesLength: number) => number,
    },
    radius: {
        func: (node: Node) => number,
    }
    linkWidth: {
        func: (link: Link) => number,
    },
    depth: number,
    animationsEnabled: boolean
}
```

And then, just by setting the `func` to a different function that can take, for example, the node and calculate the radius for that specific node, we can completely change the way the graph is being rendered. To allow the user to change the settings, we can use the features of React such as the `useState()` hook to make a reactive variables that keep the information about the user's configuration and based on these variables, the settings can be changed accordingly.

3.3.2 List of culprits

Unfortunately, nivo's network graph does not support any other interactivity with the graph other than the tooltip. A feature that would make moving from one student to another much easier is an `onClick` event that would be attached to each of the nodes and would take the teacher to the page with this user's plagiarism data. This feature is requested by the nivo's community but not yet implemented.

It would make some sense to move this link to the tooltip, but this is also not possible due to the design of the tooltip. It is impossible to hover over the tooltip and click on the link as the tooltip's position is moving along with the cursor.

This forced us to add another sub-component to the graph. The sole purpose of this component is for the teachers to quickly find the student and get taken to the page with this particular student's plagiarism data after clicking on the student. This component also supports a quick search box to make this process faster.

3.3.3 Responsivity

To make this graph component fully responsive, we can make use of the flexbox solution again, but this time it is not ideal. The problem is that the 3 components on the screen are by default sorted (from left to right) like this: `Settings - Graph - List of culprits` and after the user makes the screen smaller the last component will get separated: `Settings - Graph` and below that: `List of culprits`. This flexbox solution is very practical when dealing with lists of data that will automatically get aligned, but with just 3 components, this solution lacks the flexibility needed to create the perfect user experience.

To make the solution truly flexible, we used CSS's grid which allows us to set the areas for each of the components. To set up the grid, the components need to be wrapped in a common parent. The parent can get the class of `wrapper` to be later referenced in the CSS file.

```
.wrapper {
    grid-template-areas:
        "settings graph culprits";
    grid-template-columns:
        minmax(160px, 15%) auto minmax(160px, 15%);
    grid-template-rows:
        minmax(500, 1000);
}
```

The code above will create 3 areas named `settings`, `graph` and `culprits` and set the width of both the `settings` and the `culprits` area to be at least 160px and at most 15% of the whole width of the wrapper. The `graph` area will take up the rest of the width remaining.

In other words, both the `settings` and the `culprits` areas will be at 15% of the width of the wrapper, but if the width of the wrapper gets too small, they will not shrink under 160px, and the `graph` area will start shrinking instead.

To change the layout when the width of the browser is too small, we can use media queries that will restrict the CSS code within them to be only applicable when the width of the screen is in a certain range.

```
@media (max-width: 1500px) and (min-width: 1000px) {
    grid-template-areas:
        "settings settings"
        "graph culprits";
    grid-template-columns: auto minmax(190px, 15%);
    grid-template-rows: auto minmax(500px, 1fr);
}
```

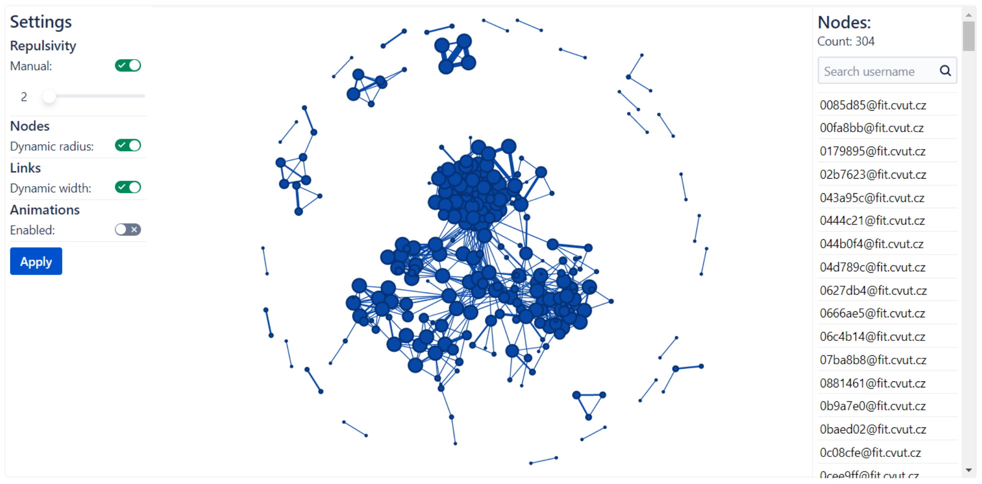
In this example, the rules inside the media query will only be used if the width of the screen is at least 1000px and at most 1500px. If this condition is true, the areas will be in two rows – `settings` area will be in the first row, and the `graph` and `culprits` areas will be in the second row.

The first row's height will be determined by the height of the content inside of it (by the content of the `Settings` component). The second's row height will be at least 500px and more.

Then by assigning the area to each of the components using the `grid-area: <name>` rule, the component will be fitted into the assigned area.

This solution is very flexible as by changing the placement of the areas in the wrapper, one can change the whole design of the component or potentially the whole page.

Because the list of culprits component can get very long, we decided to restrict the height of this component, and any elements overflowing vertically are then hidden but still accessible using the scrollbar that is specific to the component.


Settings
Repulsivity
Manual: ✓
2
Nodes
Dynamic radius: ✓
Links
Dynamic width: ✓
Animations
Enabled: ✗

Nodes:
Count: 304

0085d85@fit.cvut.cz
00fa8bb@fit.cvut.cz
0179895@fit.cvut.cz
02b7623@fit.cvut.cz
043a95c@fit.cvut.cz
0444c21@fit.cvut.cz
044b0f4@fit.cvut.cz
04d789c@fit.cvut.cz
0627db4@fit.cvut.cz
0666ae5@fit.cvut.cz
06c4b14@fit.cvut.cz
07ba8b8@fit.cvut.cz
0881461@fit.cvut.cz
0b9a7e0@fit.cvut.cz
0baed02@fit.cvut.cz
0c08cfe@fit.cvut.cz
0cee9ff@fit.cvut.cz

Figure 3.4: The final design of the network graph component.

3.4 Data export

The application also offers the option to export the data. There are 3 options to choose from:

- CSV
- JSON minified - valid JSON file that has been compressed by removing all the unnecessary information (tabs, whitespaces, etc.)
- JSON pretty - formatted JSON file to improve human-readability

The format can be selected from a dropdown selection menu, and after clicking the download button, the user will receive the file containing the data in the selected format.

By creating a reactive variable using the `useState()` hook, we can track which format the user has chosen and after `onClick` event is called on the download button, we can convert the data to the selected format and send it to the user.

It is implemented as a React component which makes this solution reusable. This component can potentially be placed in different parts of the application, beyond just the plagiarism part, to export data for the user.

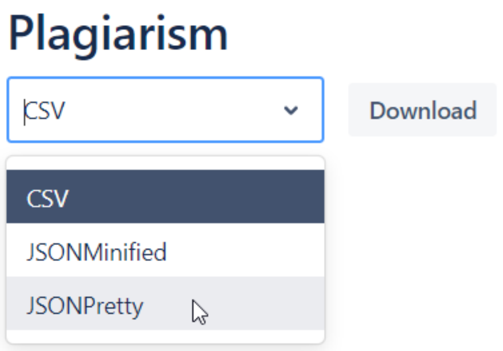


Figure 3.5: The design of the export component.

Chapter 4

Testing and documentation

"Testing is the process of executing a program with the intent of finding errors."[34] Every software has to go through at least some form of testing. The most basic form of testing is manual testing which is also the most primitive one but also used the most. Manually trying, what the program does after a certain input is given, is very time-consuming, which also demotivates developers and testers to test the functionality thoroughly enough. But there are much better ways to test software using automation and writing some additional code that will test the functionality in-depth, and it takes much less time to do so than manual testing would.

The most common types of tests are:

- Unit testing - testing individual classes, functions, ...
- Integration testing - testing interactions between components
- System testing - testing the system as a whole and if it satisfies the specified requirements

We also made sure that the code is clean and self-documenting by having a uniform naming system. The code is also commented inside the code using Python's docstring comments and using JavaScript's JSDoc but also commenting on the important parts of the code.

4.1 Unit tests

As said above, unit tests usually test a singular part of the application. The main idea of unit tests is that they should be fast and isolated. Unit tests should be ran very frequently to check that the most basic logic is still functional and works as intended. The isolation is important because if a unit test fails, the isolation of that unit test makes sure that the deficit is within that one individual part of the application, making finding and fixing the issue much quicker as the developer knows exactly in which part of the application the issue is.

4.2 Unit Testing in React using JEST

"Jest is a delightful JavaScript Testing Framework with a focus on simplicity. It works with projects using: Babel, TypeScript, Node, React, Angular, Vue and more!"[35]

As React is built by Facebook, the obvious tool for testing (not only) React applications is Jest that is also developed by Facebook. Jest offers good documentation, and there are a lot of resources as it is one of the most used tools for testing JavaScript and makes testing it very easy while providing very clear messages after the testing is done, which makes it effortless to find which part of the application is not working.

4.2.1 Fixtures

"A test fixture is a fixed state of a set of objects used as a baseline for running tests. The purpose of a test fixture is to ensure that there is a well known and fixed environment in which tests are run so that results are repeatable."[36]

To create these fixtures, we used some real data that the previous version of the LearnShell's plagiarism detection algorithm got. Keeping in mind that unit tests should be compact and also later they could be used as a secondary documentation to the code, we picked only a few records from the dataset, which makes it much easier for a human to check what the tested functions do by glancing at the input and the output.

4.2.2 Testing the utility functions

We focused mainly on testing the utility functions mainly because they contain a big majority of the logic of this work. As mentioned in previous chapters trying to write code in a more functional way is very beneficial. Functional code has no inner state, meaning that testing such code requires a lot fewer tests, and the code itself is much less prone to errors. Simply testing the input against the expected output is all one has to do to test such functions, whereas testing code that has some inner state is not as simple. Such function can change its state from one to another several times until the error in the code shows itself resulting in an unexpected result. That is why all the utility functions are **stateless** and **immutability** was a priority.

4.3 Unit testing in Django using unittest

The preferred way to test Django code is using unittest, which we ultimately decided to use because of how natural choice it is as it is a module of the Python's standard library.

Unittest uses the class-based approach to writing tests. Each test class can have many different methods that together make up the test suite to test the chosen part of the application.

We focused mainly on testing the utility functions as they have a clear input/output relation that can be easily tested. These functions also make up a big part of the logic behind the detection algorithm, so focusing on thoroughly testing, for example, the LCS algorithm was crucial to the correctness of the results returned by the detection algorithm as a whole.

Testing the code was again much more simple by adhering to the principles introduced in the previous section.

4.4 User documentation

The front-end part of this work is available to the teachers as part of the LearnShell web application with the URL suffix `/plagiarism`. This is the main page, where all the plagiarism data is displayed. Data from this page can be exported by first selecting a format and then clicking the `Download` button.

The second important page is `/plagiarism/user/[email]` – a page with plagiarism data of a user-specified by their email. This page can be accessed by clicking on a username in any component of the plagiarism module.

The `/plagiarism/assignment/[name]` can be used to display the suspicion detected in the context of a specified assignment.

4.4.1 List of plagiats

The list of plagiats displays all the culprits with a shared solution. At the bottom of every suspicion is a `Resolve` button to resolve the suspicion.

The data can be filtered by typing a name of an assignment or a username in the search field. There are also 4 sorting options in the selection menu:

- Culprits count ascending
- Culprits count descending
- Assignment name ascending
- Assignment name descending

4.4.2 Graph of culprits

The most dominant part of the page is the graph of culprits, which shows the relations between all the culprits. By default, the bigger the node is, the more cribbed solutions the student has, and the thicker the line between the nodes is, the stronger the relationship between the two culprits.

More information is accessible by hovering the cursor over a node. This will trigger the tooltip, which can be used to display the email of the culprit, the number of cribbed solutions and the depth.

The graph displays all the culprits when on the main page, but it will display only the direct coworkers when a particular student is selected. The main student will be displayed in a different color and will have a depth of 0. The direct coworkers will have a depth of 1. There is a slider in the settings area to increase this depth to show more students.

The nodes area can be used to go to the page of a particular student. Typing a part of the username in the search box can speed up searching for a specific user.

There are more settings that can be changed in the settings area.

The first setting is for enabling manual repulsivity. When manual repulsivity is enabled, a slider appears for setting the value. If the number is high, the nodes will repel each other more, resulting in the graph to be more sparse. If the number is low, the nodes will be closer to each other, and the graph will be denser. This feature is very useful when the automatic repulsivity fails to display the graph correctly, and the graph overflows and part of it is hidden.

The other two options are to disable the dynamic radius of the nodes and the dynamic thickness of the links. This is useful when nodes or links are too big and are covering up part of the graph.

The last option is to enable to see how the graph is changing, which is especially helpful when adjusting the depth. This option is recommended to be disabled when having performance issues.

It is needed to click on the `Apply` button for the changes to have any effect.

4.4.3 Tables

Any table with data can be sorted by clicking on the column. There are 3 stages that are indicated by an arrow in which the column is:

- Sorting ascending – Arrow up
- Sorting descending – Arrow down
- Not sorting – No arrow

The tables only display 10 rows at a time. The pagination below the table can be used to switch to a different page by either clicking the arrows to move by 1 page or by clicking a number to move to the specified page.

Conclusion

The goal of this work was to design and implement a working user interface for teachers to inspect, analyze and resolve plagiarism suspicions and improve the detection algorithm.

Such solution was implemented and described in this bachelor thesis. This solution is ready to be deployed in the live version of LearnShell for the next semester. Deploying it should be a very simple task as the whole solution is built on top of all the already existing LearnShell's modules. This solution works and satisfies all of the requirements. Teachers can aggregate suspicions by assignments, students or inspect all the suspicions at once. There are several useful statistics extracted from the data that teachers can display as well as a visualization using nivo's network graph for teachers to see the clusters of culprits and recognize patterns.

The detection algorithm was improved to process and assign numerical values to any two scripts. The whole detection algorithm is very flexible, and we think spending extra time on crafting such a flexible solution that can be easily expanded upon is well worth it. We believe this time is not wasted as someone will surely appreciate this flexibility in the future while improving the algorithm furthermore, using the suggestions provided in this work.

In the end, this work built up a solid foundation for any LearnShell's future improvements that can be made both on the front-end and the detection algorithm.

Bibliography

[1] University of Oxford. Plagiarism. [online], 2021, [accessed: 2021-04-21]. Available from: `https://www.ox.ac.uk/students/academic/guidance/skills/plagiarism`

[2] Streefkerk, R. Types of plagiarism. [online], [accessed: 2021-04-21]. Available from: `https://www.scribbr.com/plagiarism/types-of-plagiarism/`

[3] xenteros; Laurel. Plagiarism and using/copying code from Stack Overflow and submitting it in an assignment. [online], December 2016, [accessed: 2021-04-21]. Available from: `https://meta.stackoverflow.com/questions/339152/plagiarism-and-using-copying-code-from-stack-overflow-and-submitting-it-in-an-as`

[4] StackOverflow, Inc. Public Network Terms of Service. [online], May 2020, [accessed: 2021-04-21]. Available from: `https://stackoverflow.com/legal/terms-of-service#licensing`

[5] Commons, C. Attribution-ShareAlike 4.0 International (CC BY-SA 4.0). [online], [accessed: 2021-04-21]. Available from: `https://creativecommons.org/licenses/by-sa/4.0/`

[6] Harary, F. *Graph Theory.* Addison-Wesley, 1969, ISBN 0201027879.

[7] Abstract Syntax Tree. [online], [accessed: 2021-04-24]. Available from: `https://deepsource.io/glossary/ast/`

[8] Wikimedia Foundation, Inc. Abstract syntax tree. [online], [accessed: 2021-05-02]. Available from: `https://en.wikipedia.org/wiki/Abstract_syntax_tree`

[9] Elliott, E. *Composing Software: An Exploration of Functional Programming and Object Composition in JavaScript.* Independently Published, 2018, ISBN 9781661212568.

[10] Django Software Foundation. Meet Django. [online], [accessed: 2021-04-22]. Available from: https://www.djangoproject.com/

[11] MDN contributors. Django introduction. [online], [accessed: 2021-04-22]. Available from: https://developer.mozilla.org/en-US/docs/Learn/Server-side/Django/Introduction

[12] MDN contributors. Introduction to web APIs. [online], [accessed: 2021-04-22]. Available from: https://developer.mozilla.org/en-US/docs/Learn/JavaScript/Client-side_web_APIs/Introduction

[13] Opidi, A. GraphQL vs. REST: A Comprehensive Comparison. [online], [accessed: 2021-04-22]. Available from: https://blog.api.rakuten.net/graphql-vs-rest/

[14] Red Hat, Inc. What is a REST API? [online], [accessed: 2021-04-22]. Available from: https://www.redhat.com/en/topics/api/what-is-a-rest-api

[15] Stubailo, S. GraphQL vs. REST. [online], [accessed: 2021-04-22]. Available from: https://www.apollographql.com/blog/graphql-vs-rest-5d425123e34b/

[16] The GraphQL Foundation. A query language for your API. [online], [accessed: 2021-04-22]. Available from: https://graphql.org/

[17] The GraphQL Foundation. Introduction to GraphQL. [online], [accessed: 2021-04-22]. Available from: https://graphql.org/learn/

[18] howtographql.com. GraphQL is the better REST. [online], [accessed: 2021-04-22]. Available from: https://www.howtographql.com/basics/1-graphql-is-the-better-rest/

[19] The GraphQL Foundation. GraphQL Best Practices. [online], [accessed: 2021-04-22]. Available from: https://graphql.org/learn/best-practices/

[20] Trocki, W. GraphQL performance explained. [online], [accessed: 2021-04-22]. Available from: https://medium.com/@wtr/graphql-performance-explained-cb4b43412fb4

[21] Microsoft. What is TypeScript? [online], [accessed: 2021-04-23]. Available from: https://www.typescriptlang.org/

[22] MDN contributors. TypeScript support in Svelte. [online], [accessed: 2021-04-23]. Available from: https://developer.mozilla.org/en-US/docs/Learn/Tools_and_testing/Client-side_JavaScript_frameworks/Svelte_TypeScript

[23] MDN contributors. Getting Started. [online], [accessed: 2021-04-23]. Available from: `https://developer.mozilla.org/en-US/docs/Learn/Tools_and_testing/Client-side_JavaScript_frameworks/React_getting_started`

[24] Facebook, Inc. A JavaScript library for building user interfaces. [online], [accessed: 2021-04-23]. Available from: `https://reactjs.org/`

[25] Facebook, Inc. Getting Started. [online], [accessed: 2021-04-23]. Available from: `https://reactjs.org/docs/getting-started.html`

[26] Banks, A.; Porcello, E. *Learning React: Functional Web Development with React and Redux*. O'Reilly Media, Inc., first edition, 2017, ISBN 1491954620.

[27] Vercel, Inc. The React Framework for Production. [online], [accessed: 2021-05-01]. Available from: `https://nextjs.org/`

[28] Coper, F. *The Next.js handbook*. 2019, `https://flaviocopes.com/page/nextjs-handbook/`(visited 2021-04-20).

[29] Bostock, M. D3.js. [online], [accessed: 2021-04-20]. Available from: `https://d3js.org/`

[30] vis.js community. vis.js. [online], [accessed: 2021-04-20]. Available from: `https://visjs.org/`

[31] Benitte, R. nivo. [online], [accessed: 2021-04-20]. Available from: `https://nivo.rocks/`

[32] Ďuračík, M.; Krsak, E.; et al. Using concepts of text based plagiarism detection in source code plagiarism analysis. 2017.

[33] veksenn; zthall. Debounce. [online], [accessed: 2021-04-29]. Available from: `https://lodash.com/docs/#debounce`

[34] Myers, G. J. *The art of software testing (2. ed.)*. Wiley, 2004, ISBN 978-0-471-46912-4.

[35] Facebook, Inc. Jest. [online], 2021, [accessed: 2021-04-20]. Available from: `https://jestjs.io/`

[36] Hawks, P. Test fixtures. [online], June 2017, [accessed: 2021-04-20]. Available from: `https://github.com/junit-team/junit4/wiki/test-fixtures`

Appendix A

Acronyms

AST	Abstract syntax tree
CSV	Comma separated values
HTTP	Hypertext transfer protocol
IDE	Integrated development environment
JSON	JavaScript object notation
JSX	JavaScript XML
LCS	Longest common string
REST	Representational state transfer
TOS	Terms of service
UI	User interface
URL	Uniform resource locator
UX	User experience

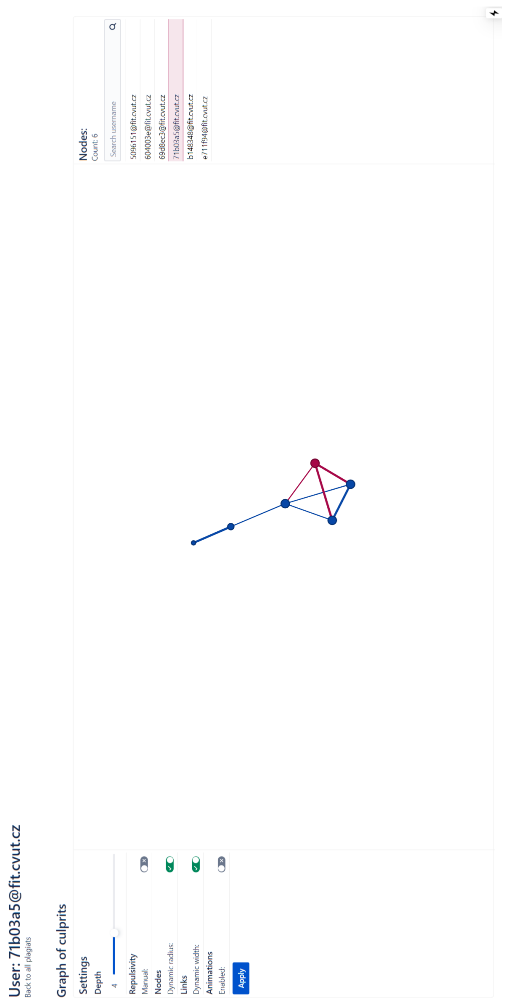
Appendix **B**

Contents of enclosed CD

```
 readme.txt ........................ the file with CD contents description
 src .................................... the directory of source codes
   thesis .............. the directory of LaTeX source codes of the thesis
 text ......................................... the thesis text directory
   thesis.pdf .......................... the thesis text in PDF format
```

Appendix C

Culprits graph with a selected culprit



User: 71b03a5@fit.cvut.cz

Back to all plagiats

Graph of culprits

Settings

Depth

4

Repulsivity

Manual:

Nodes

Dynamic radius:

Links

Dynamic width:

Animations

Enabled:

Nodes:

Count: 6

5096151@fit.cvut.cz

604003e@fit.cvut.cz

69d8ec3@fit.cvut.cz

71b03a5@fit.cvut.cz

b148348@fit.cvut.cz

e711f94@fit.cvut.cz

Appendix D

Plagiarism page

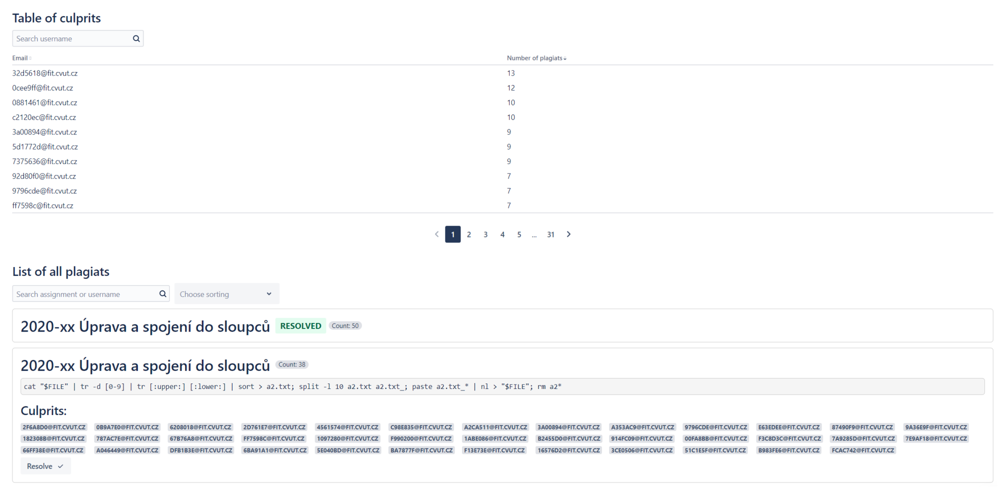


Table of culprits

Search username

Email	Number of plagiats
32d5618@fit.cvut.cz	13
0cee9ff@fit.cvut.cz	12
0881461@fit.cvut.cz	10
c2120ec@fit.cvut.cz	10
3a00894@fit.cvut.cz	9
5d1772d@fit.cvut.cz	9
7375636@fit.cvut.cz	9
92d80f0@fit.cvut.cz	7
9796cde@fit.cvut.cz	7
ff7598c@fit.cvut.cz	7

‹ 1 2 3 4 5 … 31 ›

List of all plagiats

Search assignment or username | Choose sorting

2020-xx Úprava a spojení do sloupců RESOLVED Count: 50

2020-xx Úprava a spojení do sloupců Count: 38

```
cat "$FILE" | tr -d [0-9] | tr [:upper:] [:lower:] | sort > a2.txt; split -l 10 a2.txt a2.txt_; paste a2.txt_* | nl > "$FILE"; rm a2*
```

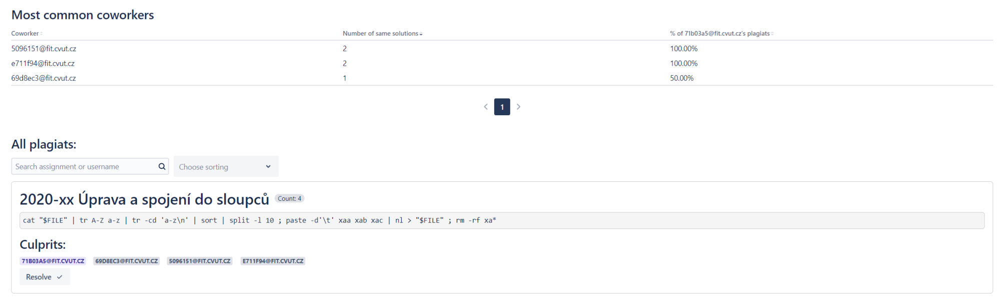
Culprits:

2F6A8D0@FIT.CVUT.CZ 0B9A7E0@FIT.CVUT.CZ 6208018@FIT.CVUT.CZ 2D761E7@FIT.CVUT.CZ 4561574@FIT.CVUT.CZ C98E835@FIT.CVUT.CZ A2CA511@FIT.CVUT.CZ 3A00894@FIT.CVUT.CZ A353AC9@FIT.CVUT.CZ 9796CDE@FIT.CVUT.CZ E63EDEE@FIT.CVUT.CZ 87490F9@FIT.CVUT.CZ 9A36E9F@FIT.CVUT.CZ
182308B@FIT.CVUT.CZ 787AC7E@FIT.CVUT.CZ 67B76A8@FIT.CVUT.CZ FF7598C@FIT.CVUT.CZ 1097280@FIT.CVUT.CZ F990200@FIT.CVUT.CZ 1ABE086@FIT.CVUT.CZ B2455D0@FIT.CVUT.CZ 914FC09@FIT.CVUT.CZ 00FA8BB@FIT.CVUT.CZ F3C8D3C@FIT.CVUT.CZ 7A9285D@FIT.CVUT.CZ 7E9AF18@FIT.CVUT.CZ
66FF38E@FIT.CVUT.CZ A046449@FIT.CVUT.CZ DFB1B3E@FIT.CVUT.CZ 6BA91A1@FIT.CVUT.CZ 5E040BD@FIT.CVUT.CZ BA7877F@FIT.CVUT.CZ F13E73E@FIT.CVUT.CZ 16576D2@FIT.CVUT.CZ 3CE0506@FIT.CVUT.CZ 51C1E5F@FIT.CVUT.CZ B983FE6@FIT.CVUT.CZ FCAC742@FIT.CVUT.CZ

Resolve ✓

Appendix E

Plagiarism page with a selected culprit



Most common coworkers

Coworker	Number of same solutions	% of 71b03a5@fit.cvut.cz's plagiats
5096151@fit.cvut.cz	2	100.00%
e711f94@fit.cvut.cz	2	100.00%
69d8ec3@fit.cvut.cz	1	50.00%

All plagiats:

2020-xx Úprava a spojení do sloupců Count: 4

```
cat "$FILE" | tr A-Z a-z | tr -cd 'a-z\n' | sort | split -l 10 ; paste -d'\t' xaa xab xac | nl > "$FILE" ; rm -rf xa*
```

Culprits:

71B03A5@FIT.CVUT.CZ 69D8EC3@FIT.CVUT.CZ 5096151@FIT.CVUT.CZ E711F94@FIT.CVUT.CZ