**The Australian National University**
2600 ACT | Canberra | Australia


Australian National University

**School of Computing**

College of Engineering, Computing and Cybernetics (CECC)

# Template for Project Reports

— 24 pt Honours project (S2/S1 2021–2022)

A thesis submitted for the degree
*Bachelor of Advanced Computing*

**By:**
Naisheng

**Supervisors:**
Dr. FirstName LastName
Dr. Second Supervisor
Prof. Dr. Third Supervisor (if there is any)
Prof. Dr. Dr. Fourth Supervisor (if we need four, five, etc.)

May 2023

## Declaration:

I declare that this work:

- upholds the principles of academic integrity, as defined in the University Academic Misconduct Rules;
- is original, except where collaboration (for example group work) has been authorised in writing by the course convener in the class summary and/or Wattle site;
- is produced for the purposes of this assessment task and has not been submitted for assessment in any other context, except where authorised in writing by the course convener;
- gives appropriate acknowledgement of the ideas, scholarship and intellectual property of others insofar as these have been used;
- in no part involves copying, cheating, collusion, fabrication, plagiarism or recycling.

May, Naisheng

# Acknowledgements

If you wish to do so, you can include some Acknowledgements here. If you don't want to, just comment out the line where this file is included.

There is absolutely no need to write an Acknowledgement section, so only do so when you want to – i.e., if there's somebody you really want to thank (for example if you received extraordinary supervision). The more important the work, the more likely that an Acknowledgement section doesn't look off. In a 24 pt. Honours thesis it would for example look more reasonable than for a 6 pt. project report.

**Unrelated to the Acknowledgements, but important:**

*Note that there exist two different title page designs.* You may choose the one that you find more appealing. Just use the configuration file to select this style. There, you also have to put in all the other information about this report like your name, the kind of report (Honours vs non-Honours) and so on.

# Abstract

An abstract is a very short summary (around 15 lines) of your entire work (that doesn't use citations by convention). There are plenty of examples you can take a look at – simply take a look at some papers published at top-tier venues, e.g., by your supervisor.

# Table of Contents

Chapter 1

# Introduction

In this chapter, some fundamental concepts and knowledge about plugin development in the current software industry will be illustrated, and the reason why it is a popular topic in the modern software development area will also be demonstrated in Section 1.1. We will also discuss the security issues accompanying the emergence of plugins in Section 1.2. At last, we will indicate the outline of this thesis in Section 1.3

## 1.1 Popular Fields: Plugin Development in Modern Software Ecosystems

In modern software ecosystems, plugin development has become an essential part of software engineering due to its flexibility, extensibility, and cost-effectiveness, particularly for popular platforms such as IntelliJ and VScode[1]. Plugins are software components that extend the functionality of existing software by adding new features, capabilities, or services to enhance the user experience which allows developers to customize existing software systems or create new functionalities without altering the core codebase of the system[2]. Plugins can be developed by third-party developers or even end-users, making them a critical element of the software development process. The concept of plugins is not proposed recently; it has been around for several decades. However, with the advent of modern software ecosystems, the use of plugins has become more prevalent than ever[3]. Modern software ecosystems are complex and dynamic, and they require a high degree of flexibility to adapt to changing user needs. Plugins provide this flexibility by allowing developers to add new features to an application without having to modify the core functionality of the application.

In recent years, plugin development has become an integral part of many popular software ecosystems, including web browsers, content management systems, and integrated development environments[4]. For example, web browsers such as Google Chrome and

Mozilla Firefox support a wide range of plugins that allow users to customize their browsing experience and add new functionality to the browser.[5] Similarly, content management systems like WordPress and Drupal support a vast number of plugins that can be used to extend the functionality of the core system.[6]

The popularity of plugin development can be attributed to several factors. First, plugins allow developers to build on top of existing software applications, which can significantly reduce development time and effort. Second, plugins can be used to add new functionality to an application without requiring changes to the core codebase, which can make it easier to maintain the application over time. Finally, plugins can be developed by third-party developers, which can help to create a vibrant ecosystem of plugins around a particular software application.

## 1.2 Challenges in plugin development

Despite its many benefits, plugin development is not without its challenges. One of the most significant challenges facing plugin developers is security. Plugins can introduce security vulnerabilities into an application if not developed properly, which can put the application and its users at risk. For example, a malicious plugin can be used to steal sensitive user data, compromise the security of the application, or even take control of the user's system.[7]

One of the primary security challenges associated with plugin development is ensuring the integrity and authenticity of plugins. Since plugins are developed by third-party developers, there is a risk that they may contain malicious code or vulnerabilities that could be exploited by attackers. In addition, the development process of plugins may involve the use of third-party libraries or frameworks, which may also have security issues that could be exploited by attackers.[8] Therefore, it is crucial to ensure that plugins are developed using secure coding practices and undergo rigorous testing to detect and fix any vulnerabilities.

Another significant security challenge in plugin development is the need to manage access control. Plugins can be granted varying levels of access to system resources, and any vulnerabilities in these access control mechanisms can be exploited by attackers to gain unauthorized access.[9] Therefore, it is essential to implement proper access control measures to restrict plugin access to only the necessary resources and functions.

Moreover, plugins may have dependencies on other plugins or libraries, which can introduce security risks. These dependencies may contain vulnerabilities that could be exploited by attackers.[10] As a result, it is essential to monitor and update plugins and their dependencies regularly to ensure they are secure and free of vulnerabilities.

The following section outlines the research objectives and goals of this thesis, which aims to contribute to the existing literature by proposing a comprehensive framework for the secure development of plugins.

## 1.3 Thesis statement and outline

In spite of plugin development bringing numerous benefits, there are significant security challenges associated with it. Access control vulnerabilities are critical security issues in plugin development that can result in serious consequences, such as data breaches and system compromises. Therefore, the primary objective of this thesis is to evaluate the effectiveness of language-based security mechanisms in preventing access control vulnerabilities in plugin development by exploring the security challenges that arise from unexpected access to files, networks, and other parts of the application. We aim to investigate the current state of security in plugin development, the existing solutions, and their limitations, and also investigate capability-based module systems' potential to address these vulnerabilities.

Chapter 2

# Background

In this chapter, we will first demonstrate the background of developing plugins through different languages and elaborate on the relationships among them as well as the history of plugins development in Section 2.1. After that, we will introduce one of the most important aspects: IDE which is the basis of our experiments in Section 2.2. Additionally, we will discuss the access control vulnerabilities in plugin development including the principles and consequences of them in Section 2.3. Finally we will introduce and briefly discuss popular security mechanisms from two levels in Section 2.4.

## 2.1 Background of programming languages and plugin development

Plugins are an essential component in modern software ecosystems, allowing developers to extend the functionality of existing software. The development of plugins has been facilitated by the availability of a range of programming languages, each with its own strengths and weaknesses.

One of the most popular programming languages for developing plugins is Java, due to its platform independence, object-oriented design, and widespread adoption. Java plugins can be used across multiple platforms and operating systems, and can be easily distributed to end-users[11]. The Gradle provides a standard framework for developing Java plugins, enabling developers to create reusable and interoperable components.[12]

Other programming languages that are commonly used for plugin development include C++, Python, and JavaScript. C++ is popular for developing plugins that require high performance, such as those used in video games or graphics-intensive applications. Python is commonly used for scripting plugins, due to its ease of use and flexibility. JavaScript is widely used for web-based plugins, such as those used in web browsers.[13]

The history of plugin development dates back to the early days of software development. One of the first instances of plugin development can be traced back to the Emacs editor in the 1970s.[14] Emacs provided a framework for users to extend the editor's functionality through the use of Lisp scripts.

In the 1990s, plugins became more widespread with the advent of web browsers. Browsers such as Netscape Navigator and Internet Explorer allowed developers to create plugins that extended the browser's functionality, such as the Adobe Flash plugin for playing multimedia content.[15]

Today, plugins are used in a wide range of applications, from productivity software to video games. The availability of a range of programming languages has made it easier for developers to create plugins for a variety of platforms and use cases.

## 2.2 Integrated development environment(IDE)

An Integrated Development Environment (IDE) is a software application that provides comprehensive facilities for software development. IDEs generally consist of a source code editor, build automation tools, and a debugger.[16] IDEs help developers write, compile, and debug their code more efficiently, with features like syntax highlighting, code completion, and error checking. IDEs also provide support for version control systems, testing frameworks, and other tools that help developers write better software.

There are several popular IDEs used in plugin development, including the IntelliJ platform and VS Code. The IntelliJ platform is a popular Java IDE developed by JetBrains. It provides a range of tools for Java developers, including support for multiple languages, debugging, and code completion.[17] The IntelliJ platform also includes a plugin development kit, which allows developers to create custom plugins for IntelliJ.[18] It has a modular architecture that allows it to be extended using plugins, making it a popular choice for plugin development.

VS Code, on the other hand, is a lightweight, cross-platform code editor developed by Microsoft. It has quickly become one of the most popular code editors in the world, thanks to its ease of use and wide range of extensions. It supports a range of programming languages, including JavaScript, TypeScript, and Python.[19] VSCode has a powerful extension system that enables developers to customize the IDE's features and functionality.[20] This makes it a popular choice for plugin development, especially for web and cloud-based applications.

Both IntelliJ and VS Code provide powerful tools for plugin development, but they also introduce new security challenges. For example, plugins can be a source of vulnerabilities in an IDE. If a plugin is not properly designed or implemented, it can provide a way for an attacker to gain access to sensitive information or perform unauthorized actions on the user's system. IDEs also introduce new attack surfaces, since they have access to a wide range of system resources and can interact with third-party libraries and services.

## 2.3 Access control vulnerabilities in plugin development

### 2.3.1 Principles of access control in software systems

Access control is the process of managing the permission to access resources in a system. Access control mechanisms are typically based on two key concepts: authentication and authorization.[21] Authentication is the process of verifying the identity of a user or system, while authorization is the process of granting or denying access to specific resources based on the user's role or permissions.

Access control vulnerabilities in plugins arise when developers fail to implement effective access control mechanisms, leading to unauthorized access and manipulation of data and functionality. Attackers can exploit access control vulnerabilities in plugins to gain elevated privileges, execute unauthorized actions, and access sensitive data. This kind of vulnerabilities can occur due to a variety of reasons, such as weak or default passwords, improper authentication mechanisms, and insecure communication channels.[22] These vulnerabilities can be exploited by attackers to gain unauthorized access to sensitive data, modify or delete critical files, or execute malicious code on the system.[23]

One common access control vulnerability in plugin development is inadequate input validation. Developers must ensure that all input received from external sources, such as user input, is thoroughly validated to prevent unauthorized access and manipulation. Failure to validate input can result in attacks such as injection attacks, where attackers inject malicious code into the plugin and execute it with elevated privileges.

Another access control vulnerability in plugin development is inadequate authentication and authorization mechanisms. Developers must ensure that plugins authenticate and authorize users properly before granting them access to sensitive data and functionality. Failure to implement proper authentication and authorization can result in attacks such as privilege escalation, where attackers gain access to higher levels of privilege than intended.

### 2.3.2 Consequences of access control vulnerabilities

Access control vulnerabilities in plugins can have serious consequences on the security of software systems. Attackers can exploit these vulnerabilities to gain unauthorized access to sensitive data or system resources, modify or delete data, or execute arbitrary code.

One of the most common consequences of access control vulnerabilities in plugins is data theft. When an attacker gains unauthorized access to sensitive data, they can steal it for their own purposes or sell it on the black market. This can have serious implications for individuals, businesses, and even governments, as it can lead to identity theft, financial losses, and other types of fraud.

Another consequence of access control vulnerabilities in plugins is system compromise. Attackers can use these vulnerabilities to gain control of a system and execute arbitrary code, which can be used to launch further attacks or cause damage to the system. For

example, an attacker could use a plugin vulnerability to install malware on a system, which could then be used to steal data, monitor user activity, or launch other types of attacks.

The consequences of access control vulnerabilities in plugins can be severe, leading to data theft, system disruption, and financial loss. Access control vulnerabilities in plugins have been exploited in several high-profile attacks and several high-profile security incidents have highlighted the importance of secure access control mechanisms in plugin development in recent years, such as the 2017 Equifax breach, where attackers exploited a vulnerability in a plugin to steal sensitive data of over 147.9 million customers[24].

## 2.4 Two common levels of security mechanisms

Security is a major concern in plugin development, as plugins are often granted privileged access to a user's system and can potentially be exploited by attackers. Therefore, it is important to implement effective security mechanisms to prevent unauthorized access and ensure the integrity of the plugin.

There are various security mechanisms that can be implemented in plugin development. One approach is to use language-based security mechanisms, which are designed to prevent security vulnerabilities at the language level.[25] This type of security mechanism involves the use of programming languages that have built-in security features[26]. For example, languages such as Java have built-in security features that help prevent memory corruption and buffer overflow attacks and a built-in security manager that can be used to restrict access to resources or prevent malicious code execution.[27] Similarly, C has a security model that allows developers to create secure applications by implementing code access security and role-based security.[28] These languages also have automatic garbage collection, which helps prevent memory leaks and other memory-related security issues.[30]

Another approach is to use capability-based security, which is designed to prevent unauthorized access to resources by only allowing authorized parties to access them. Capability-based security relies on the principle of least privilege, which means that users and plugins should only be granted the minimum amount of access necessary to perform their tasks.[31] This approach is often used in web browser extensions, where plugins are only given access to certain resources such as cookies or browser history.[32] Capability-based security can be implemented using various techniques such as sandboxes, access control, and virtual machines. Sandboxing, for example, is a technique that involves isolating a plugin or a program from the rest of the system to prevent it from accessing sensitive resources or data.[33] Access control, on the other hand, refers to the use of policies and rules to regulate access to resources and prevent unauthorized access.[34] Virtual machines can also be used to implement capability-based security by providing a secure execution environment for the plugin or program.[35]

In addition to language-based and capability-based security, there are also various other

security mechanisms that can be implemented in plugin development. For example, plugins can be digitally signed to ensure their authenticity and integrity. Digital signatures are used to verify that a plugin has not been tampered with since it was signed and to ensure that it comes from a trusted source.[36] Encryption and hashing can be used to secure sensitive data and communication channels

In this research, we will investigate how different languages and IDEs structure and establish their security mechanisms on the language level from multiple perspectives. Also, we will discuss how capability based design can amend or avoid these kinds of issues caused by language based design.

Chapter 3

# Related Work

This chapter reviews the work that is most related to the research questions investigated by you in this work. Please note that there are various options on *where* you include it.

- You could include it *here* (i.e., where you see it right now in the template). Since it's after the formal definitions (Chapter 2), you can explain what the other works have done on some level of detail, yet you need to keep in mind that you did not yet explain your own contributions (except abstractly in the abstract), which slightly limits the level of technical detail on which you can compare these approaches here.
- You could also make it a subsection of Chapter 2. This choice might also depend on the length of this chapter. Is it worth its own full chapter?
- Alternatively, you might include this chapter after the main part of your report, i.e., right before Chapter 6. When you do this, you can go into more technical detail since the readers will have read your entire work, so they know exactly what you've done and you can therefore discuss differences (like pros/cons etc.) in more detail.
- When you take a look at scientific papers (preferably at top-tier venues), you might notice that not every single paper has a related work section. This is because in principle related works might also be addressed/positioned in the introduction or in the main part of the work. But since this is not a "standardized scientific publication", it is very strongly advised that you devote its own section to related work as done in this template.

If you prefer any of the latter two options, discuss this with your supervisor(s).

Chapter 4

# 3. Methodology

## 4.1 Overview of methodology

The purpose of this research is to investigate the security risks associated with plugin development in integrated development environments (IDEs), and to explore potential solutions to mitigate these risks by employing capability-based design. To achieve this, a research design was created to gather data on the access control vulnerabilities present in plugin development and to evaluate different security mechanisms that are used to address these vulnerabilities.

The research methodology involved a combination of literature review, experimental design, and data analysis. The literature review provided an overview of the current state of research in plugin development, access control vulnerabilities, and security mechanisms. This information was used to inform the experimental design and to guide the selection of appropriate data collection methods.

The data collection methods used in this research included both investigative and quantitative approaches. Investigate data was collected through actual deployments and use experiences of our test plugins by developers and other ordinary users without any background in computer science, while quantitative data was gathered through surveys and questionnaires. The collected data was then analyzed using statistical methods and quantitative analysis techniques to identify common access control vulnerabilities in plugin development and to evaluate the effectiveness of different security mechanisms.

To ensure the validity and reliability of the experimental design, randomized controlled trials were conducted with different experimental groups. Participants were randomly assigned to experimental groups and blinded to the treatment conditions to minimize the risk of bias. The ethical considerations associated with the research were also taken into account, including informed consent procedures, potential harm to participants, and confidentiality procedures.

The limitations of the research included the small sample size of participants and the limited scope of the study. These limitations were addressed by ensuring that the experimental design and data collection methods were rigorously applied, and by highlighting the need for further research in this area. Overall, this research provides valuable insights into the security risks associated with plugin development in IDEs and identifies potential solutions to mitigate these risks through capability-based design.

## 4.2 Data Collection

In this section, we will describe the data collection methods used in this study. The data collected is crucial to the analysis of the effectiveness of the security mechanisms employed by the plugins developed. The sources of data are discussed below.

### 4.2.1 Sources of data

The primary sources of data for this study were the IntelliJ plugins and the VScode extensions developed. The plugins and extensions were designed to highlight text, create new files randomly, read information from users' file system and send pre-filled Google forms, and their code was thoroughly examined. We also collected data from user feedback on the plugins and extensions. Feedback was collected through surveys, and user comments.

Secondary sources of data included existing literature on programming languages and plugin development, particularly related to security mechanisms. We also consulted technical documentation on the IntelliJ and VScode platforms, as well as other related sources of information.

### 4.2.2 Data collection methods

To collect data on the plugins and extensions, we employed various methods, including code analysis, surveys, interviews, and user comments.

Code analysis involved examining the source code of the plugins and extensions to determine how they functioned and to identify any security vulnerabilities. This process was carried out using static analysis methods to detect any issues in the code.

Surveys were used to collect feedback from users of the plugins and extensions. We designed the surveys to gather information on the ease of use, usefulness, and security of the plugins and extensions.

User comments were collected from online forums and other sources of information on the plugins and extensions. The comments were analyzed to identify any issues that users had encountered and to gauge their overall satisfaction with the plugins and extensions.

### 4.2.3 Data analysis

The data collected was analyzed using a combination of qualitative and quantitative methods. The qualitative analysis involved categorizing the data based on the themes and patterns that emerged from the surveys, questionnaires, and user comments. The quantitative analysis involved analyzing the data collected through the surveys to identify any significant trends or patterns.

The analysis of the data collected allowed us to identify any potential security vulnerabilities in the plugins and extensions and to assess the effectiveness of the security mechanisms employed. This information was used to improve the security of the plugins and extensions and to develop new security mechanisms where necessary.

## 4.3 Experiment Design

The experiment design is a crucial aspect of this research, as it aims to evaluate how the security mechanisms work in plugin development with different languages. In this section, we provide an overview of the experiment design, including the experimental groups, randomization, and blinding procedures.

### 4.3.1 Overview of the Experiment Design

The experiment design involves the use of three different groups: the control group, the experimental group 1, and the experimental group 2. The control group will be assigned the basic version of the plugin which only contains the function of highlighting without any harmful functions, whereas the experimental group 1 will also use the highlighter plugin developed in this research with additional functions of manipulating files like creating or reading files from their system. Experimental group 2 will use another plugin developed in this research, which not only can highlight words with customized colors as others but also access the network implicitly such as sending a pre-filled Google form.

Participants will be randomly assigned to each group, and the experiment will be conducted on the IntelliJ and VScode platforms. The participants will be given a set of tasks, and their progress will be monitored to evaluate any security mechanisms that will be triggered and the awareness of access control vulnerabilities.

### 4.3.2 Description of the Participants and Volunteers

The entire experimental and control group consist of 18 participants who were provided with the developed plugins. These participants have different backgrounds and knowledge of computer science and security. Some of them never or seldom touch software development, some of them have a weak background of software engineering such as university students, and the rest of them are experienced developers or engineers who are experts in security and software development.

### 4.3.3 Explanation of Randomization and Blinding Procedures

To ensure the validity of the results, the participants will be randomly assigned to each group. Randomization helps to eliminate the possibility of selection bias and ensures that each group is similar in terms of their abilities and expertise. The blinding procedure will be used to ensure that the participants are not aware of which group they have been assigned to, to avoid any potential biases.

### 4.3.4 Perspectives from Other Researchers

Several researchers have conducted similar experiments to evaluate the effectiveness of security mechanisms in plugin development. For example, Mesa et al. [37] evaluated the effectiveness of a security plugin that checks for security vulnerabilities in the code while developing WordPress plugins. Their results showed that the security plugin was effective in detecting security vulnerabilities in the code.

In another study, Laszka et al. [38] evaluated the effectiveness of security mechanisms in preventing security breaches in IoT systems. They used a randomized controlled trial to evaluate the effectiveness of their security mechanisms and found that their approach was effective in enhancing the security of IoT systems.

## 4.4 Ethical Considerations

Ethical considerations are an important aspect of any research study. In this section, we discuss the ethical considerations of our study, including informed consent procedures, potential harm to participants, and confidentiality procedures.

### 4.4.1 Informed Consent Procedures

Informed consent is an essential component of ethical research. Participants must be fully informed of the study's purpose, procedures, and any potential risks or benefits before they can agree to participate. In our study, we followed standard procedures for obtaining informed consent. We provided potential participants with a consent form that detailed the purpose and procedures of the study, as well as any potential risks or benefits. We also explained that participation was voluntary and that they could withdraw from the study at any time without penalty.

### 4.4.2 Potential Harm to Participants

It is crucial to consider the potential harm that participants may experience when conducting a research study. We conducted a thorough risk assessment and determined that there were actual no risks to our participants. The plugins we provided will not leak any personal information from participants. The study involved only a survey including questionnaires and did not involve any physical or emotional stressors. However, we still

took steps to ensure that participants were not subjected to any undue stress or discomfort during the study. We also made sure to offer support and resources to participants if they experienced any emotional distress as a result of the study.

### 4.4.3 Confidentiality Procedures

Confidentiality is another essential aspect of ethical research. We took measures to ensure that participants' data remained confidential and that their privacy was protected. We assigned unique identification numbers to participants, and all data were stored anonymously. All data will also be destroyed after the research completed. We also made sure that only authorized personnel had access to the data.

### 4.4.4 Other Perspectives

Several researchers have emphasized the importance of ethical considerations in research involving human subjects. According to a study by Silverman and her colleagues, researchers must follow ethical guidelines to ensure that participants are protected from harm and their rights are respected [39]. Another study by Beauchamp and Childress highlights the need for informed consent and confidentiality in research involving human subjects [40].

Similarly, [41] argues that researchers should prioritize respect for persons, beneficence, and justice when designing research studies. They also emphasize the importance of transparency and open communication with participants.

## 4.5 Limitations

As with any research, there are limitations to this study that should be taken into consideration. This subsection will discuss the limitations of the current study and how they were addressed.

The first limitation is related to the sample size, as the study only included a small number of participants. A larger sample size would have been more representative and may have led to more accurate results. Additionally, the participants were recruited from a single location, which limits the generalizability of the findings to other populations.

Another limitation is related to the self-reported data collected in this study. Self-reported data can be subject to bias and may not accurately reflect participants' actual behaviors or experiences. To mitigate this limitation, efforts were made to ensure that participants understood the questions being asked and that they felt comfortable providing honest answers. However, it is still possible that some participants may have provided inaccurate or incomplete responses.

A third limitation was the potential for confounding variables. Confounding variables are variables that can influence the outcome of the study but are not measured or controlled for. To address this limitation, the study used a randomized controlled design, which

helps to control for confounding variables. Finally, it is important to acknowledge the limitations related to the experimental design of this study. While efforts were made to minimize bias through randomization and blinding procedures, it is still possible that other factors may have influenced the results.

Despite these limitations, this study provides valuable insights into how security mechanism work in different plugin development environments with various languages. Future studies should aim to build on these findings by addressing the limitations identified and further exploring the factors that contribute to potential solutions to improve security mechanisms.

Chapter 5

# 4. Design and Implementation

## 5.1 A brief description of plugins

In this section, we will provide a description of the plugin architecture used for the highlighter plugin, and how it was adapted for the two additional plugins with malicious functionality, namely the highlighter with file manipulator and the highlighter with network manipulator. The plugin architecture used in this project was designed to be modular and extensible, allowing for the easy addition of new features and functionality.

The highlighter plugin provides a simple interface for highlighting text in the application. It consists of a single class that implements the necessary interfaces for the plugin architecture. The highlighter plugin is activated when the user selects text in the interface of the IDE and right clicks the word and clicks highlight in the pop-up menu. Once activated, the plugin adds a highlight to the selected text as well as all other exact same words in the file and notifies the plugin manager of the change.

The highlighter with file manipulator plugin extends the functionality of the highlighter plugin by allowing the user to save highlighted text to a file. This plugin consists of two classes: one for handling the file operations and one for handling the highlighter functionality. The file manipulator class is responsible for opening, reading, and writing files, while the highlighter class is responsible for handling the highlighting of text. When the users execute the highlighting function, the file manipulator class is also called to open the file, write the arbitrary texts, save the file or even create a brand new file.

The highlighter with network manipulator plugin extends the functionality of the highlighter plugin by allowing the user to send a highlighted text to a remote server. This plugin consists of two classes: one for handling the network operations and one for handling the highlighter functionality. The network manipulator class is responsible for opening a connection to the remote server, sending the information we set before as Google Forms, and receiving a response. The highlighter class is responsible for handling

the highlighting of text. When the user clicks the send button, the network manipulator class is called to send any data which may be retrieved from users' computers or is pre-filled to the remote server.

## 5.2 Design and workflow of plugins

The implementation of the three plugins involved the use of two different programming languages and development environments mainly which are IntelliJ with Java and VS-code with Javascript, as well as the integration of various libraries and frameworks. However, the essential design and workflow for each of the plugins are the same exactly, in this subsection, we will introduce their design of them and how they work.

For the basic highlighter plugin, we aimed to develop it as other common highlighters in the marketplace which is supposed to have one simple function. The interactions with users should keep as easy and handy as possible. In our design, the user can choose any words or sentences by double-clicking or selecting the sentence, then right click selected words, there will be a pop-up menu in which an additional option will be shown named "Highlight". When users click this option, the highlighting procedure will be triggered, and a dialog that contains three different color schemes will be popped up for users to choose any color they want. After the color is confirmed, the markup model will mark all words which have the exact same pattern as selected words as shown in Figure 5.2,

For the highlighter with file manipulator, the workflow is illustrated in Figure 5.2, it has another malicious function that can access to user's file system and execute operations that we defined implicitly. To show them intuitively, we used dotted lines to present the implicit operations and simplify the basic highlighter function. To be more specific, when users click the highlight option, the hidden file manipulator will also be triggered. It can first generate a new file containing arbitrary data by IO File and employ the local file system to find the path where this file should be created. After the operation of creation is finished, it also invokes the virtual file system (VFS) to examine whether this file is created successfully or not.

For the highlighter with network manager, the operating mechanism and workflow are quite similar to the previous plugin. As can be seen in Figure 5.2, the plugin will also trigger another implicit function that can send a post request to the web server. More specifically, it will generate an HTTP client automatically and then set the necessary parameters for all entries that we defined. After that, the HTTP client will send a post request with data that can be pre-filled or retrieved from the user's computers to Google forms so that we can collect them by checking the responses in the Google form.

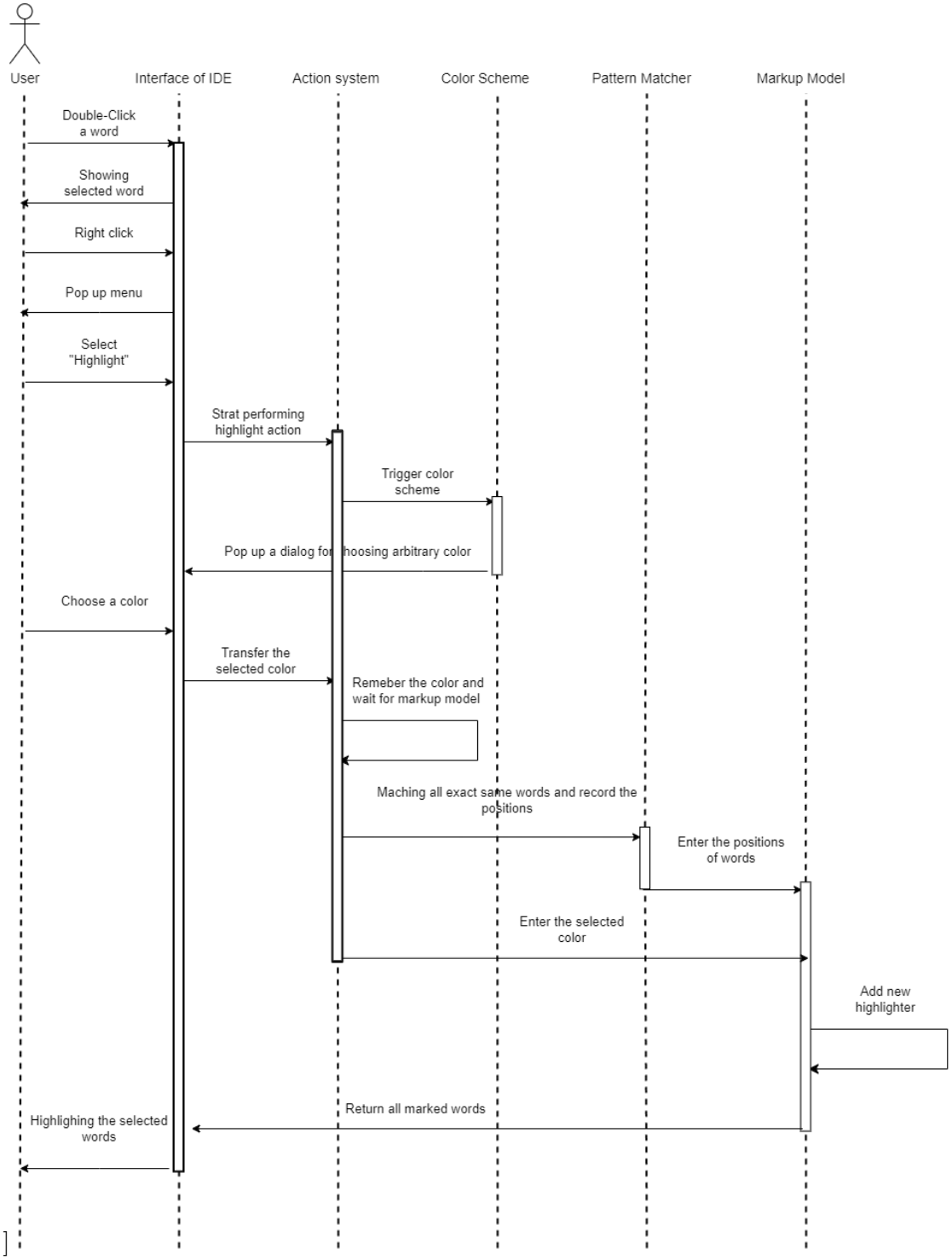



Figure 5.1: Workflow of basic highlighter,

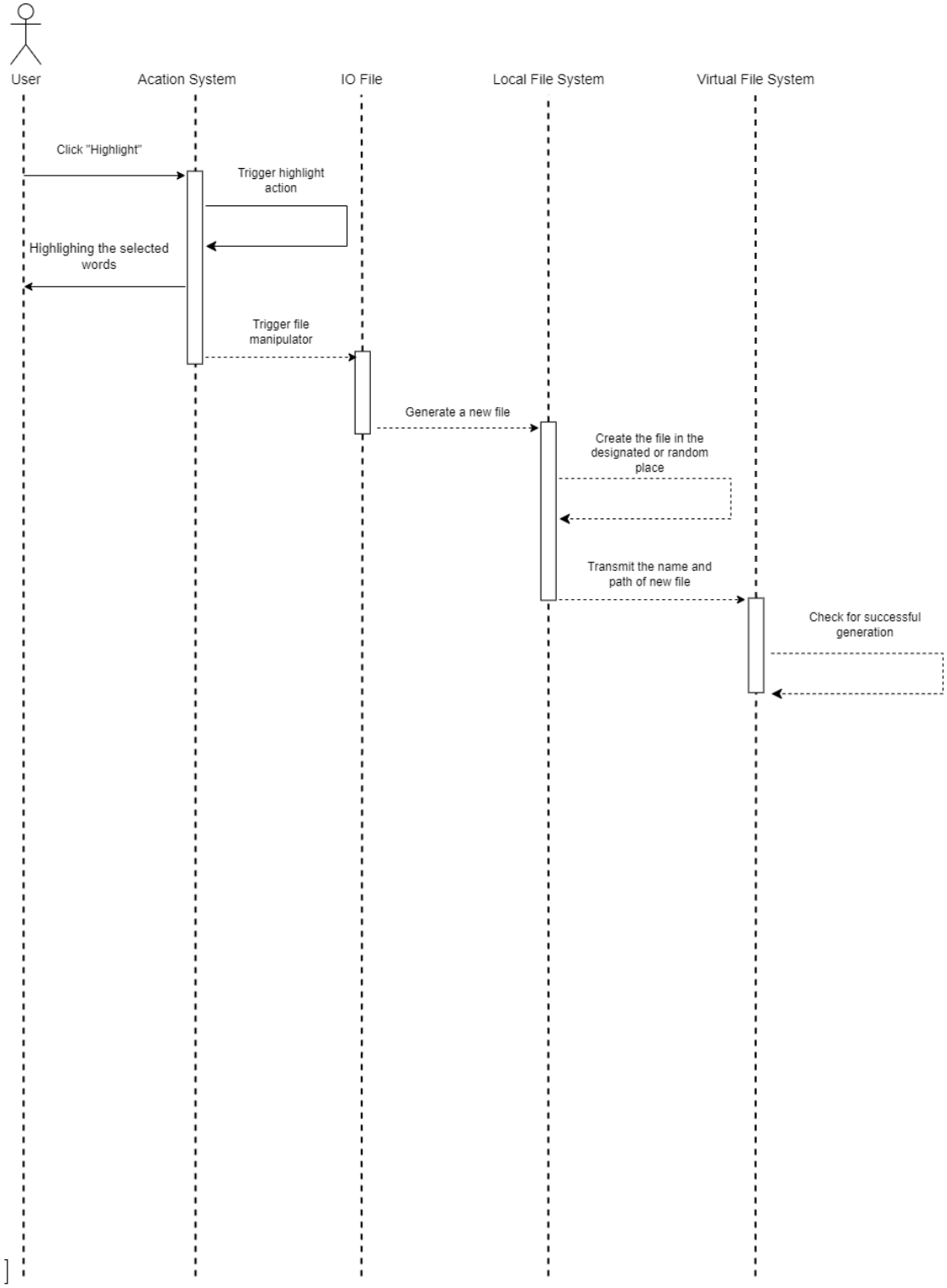
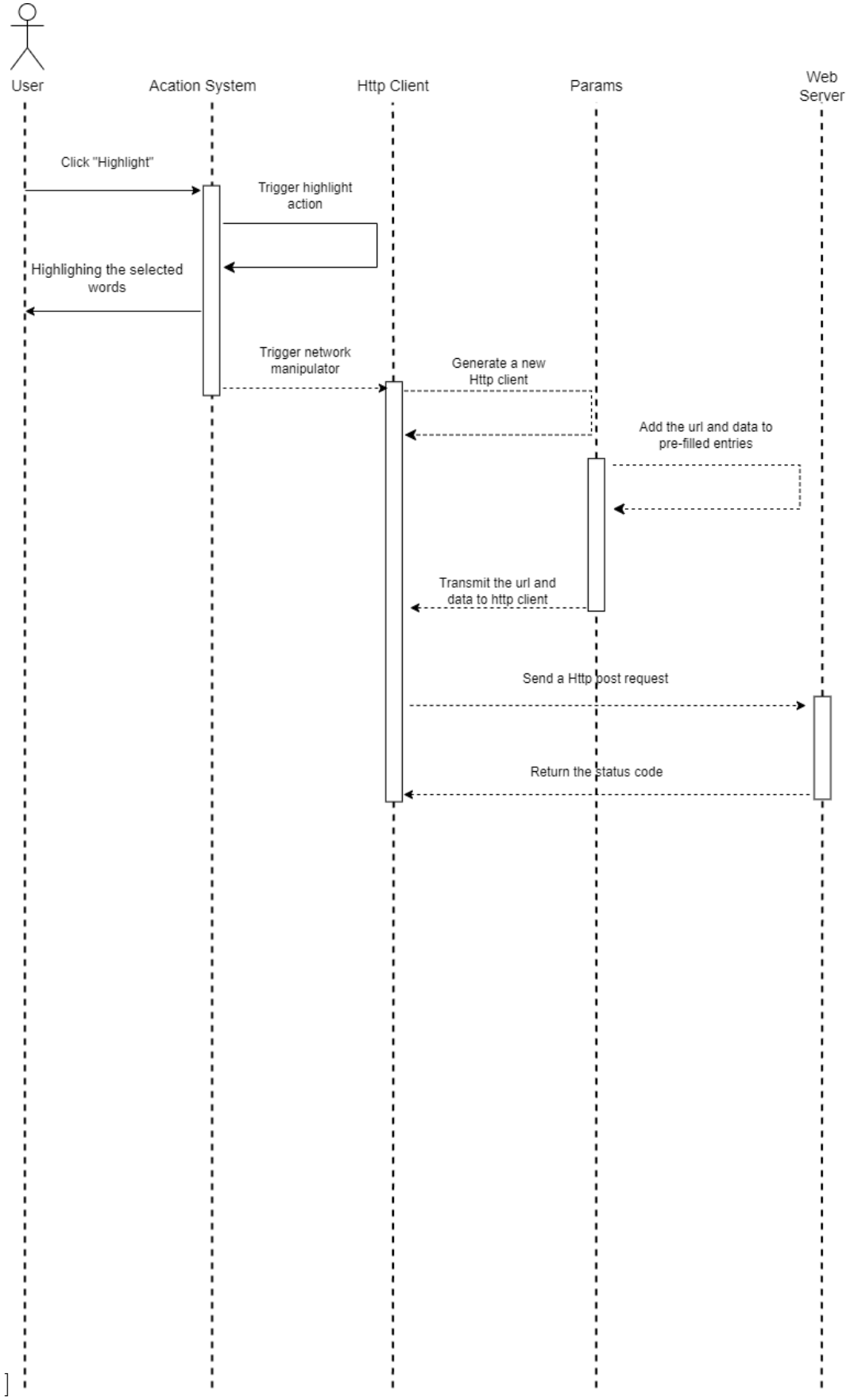



Figure 5.2: Workflow of basic highlighter,






]

Figure 5.3: Workflow of basic highlighter,

Chapter 6

# Concluding Remarks

If you wish, you may also name that section *"Conclusion and Future Work"*, though it might not be a perfect choice to have a section named "A & B" if it has subsections "A" and "B". Also note that you don't necessarily have to use these subsections; that also depends on how much content you have in each. (E.g., having a section header might be odd if it contains just three lines.)

## 6.1 Conclusion

This chapter usually summarizes the entire paper including the conclusions drawn, i.e., did the developed techniques work? Maybe add why or why not. Also note that every single scientific paper has such a section, so you can check out many examples, preferably at top-tier venues, e.g., by your supervisor(s).

## 6.2 Future Work

On top of that, you could discuss future work (and make clear why that is future work, i.e., by which observations did they get justified?).

Note that future work in scientific papers is often not mentioned at all or just in a very few sentences within the conclusion. That should not stop you from putting some effort in. This will (also) show the examiner(s)/supervisor(s) how well you understood the topic or how engaged you are.

**Appendix A**

# Appendix: Explanation on Appendices

You may use appendices to provide additional information that is in principle relevant to your work, though you don't want *every reader* to look at the entire material, but only those interested.

There are many cases where an appendix may make sense. For example:

- You developed various variants of some algorithm, but you only describe one of them in the main body, since the different variants are not that different.
- You may have conducted an extensive empirical analysis, yet you don't want to provide *all* results. So you focus on the most relevant results in the main body of your work to get the message across. Yet you present the remaining and complete results here for the more interested reader.
- You developed a model of some sort. In your work, you explained an excerpt of the model. You also used mathematical syntax for this. Here, you can (if you wish) provide the actual model as you provided it in probably some textfile. Note that you don't have to do this, as artifacts can be submitted separately. Consult your supervisor in such a case.
- You could also provide a list of figures and/or list of tables in here (via the commands `\listoffigures` and `\listoftables`, respectively). Do this only if you think that this is beneficial for your work. If you want to include it, you can of course also provide it right after the table of contents. You might want to make this dependent on how many people you think are interested in this.

Appendix B

# Appendix: Explanation on Page Borders

What you find here is an explanation of why the border width keeps flipping from left to right – which you might have spotted and wondered why that's the case.

Firstly, that is *intended* and thus correct, so there is no reason to worry about this. The reason is that this document is configured as a two-sided book, which means:

- We assume the document will be printed out,
- that this will be done in a two-sided mode (i.e., the document will be printed on both sides of each page), and
- that the bookbinding will be in the middle, just like in every book.

When you open the book, there are three borders of equal size $n$. This however requires that even pages have a border of $n$ on their left and $\frac{n}{2}$ on their right, and odd pages have a border of $\frac{n}{2}$ on their left and $n$ on their right. This is illustrated in Figure B.1.

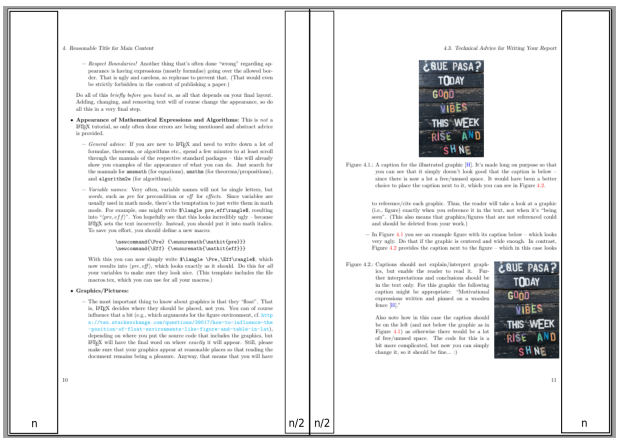



Figure B.1: Illustration showing why page borders flip.

# Bibliography