Write Right: A tool to help students write proper emails

Angus Parsons



4th Year Project Report Software Engineering School of Informatics University of Edinburgh

2023

Abstract

This report details the design, implementation, and testing of a tool, "Write Right" that helps students write emails to teachers. Write Right is a Thunderbird Extension that allows students to enter just the essential details and then automatically converts to a well-formed email. The tool also allows students to create and upload their own custom templates.

Research Ethics Approval

This project obtained approval from the Informatics Research Ethics committee. Ethics application number: 422851 Date when approval was obtained: 2022-11-22

The participant's information sheet and a consent form are included in the appendix.

Declaration

I declare that this thesis was composed by myself, that the work contained herein is my own except where explicitly stated otherwise in the text, and that this work has not been submitted for any other degree or professional qualification except as specified.

(Angus Parsons)

Acknowledgements

Special thanks to:

Brian Mitchell My supervisor for this project, offered great advice and feedback which drove the project forwards and also came up with the idea that I enjoyed implementing so much.

My friends and family I am hugely grateful for the support I have constantly felt throughout not just this year but my four years of study at The University of Edinburgh. I could not have completed this degree without them supporting me through it.



Figure 1: Project overview.

Table of Contents

1	Intr	oduction	1
	1.1	The importance of writing appropriate emails	1
	1.2	Difficulties of making Write Right	2
	1.3	Project parameters and limitations	3
2	Bac	kground	4
	2.1	Parsing text	4
		2.1.1 Parse tool choices	4
		2.1.2 PEG.js	5
	2.2	Technical background	6
		2.2.1 Thunderbird	6
	2.3	Alternatives	8
		2.3.1 ChatGPT	8
		2.3.2 Other Thunderbird extensions	8
3	Desi	gn	11
	3.1	System design	11
		3.1.1 Design method	11
		3.1.2 System requirements	12
	3.2	User interface	13
		3.2.1 Extension layouts	13
		3.2.2 Overall aesthetic	14
		3.2.3 Problems with modal dialogs	14
	3.3	System architecture	15
4	Imp	lementation	17
	4.1	Implementation method	17
	4.2	Contact detail section	17
	4.3	Manual entry section	19
	4.4	Text entry section	19
		4.4.1 Parsing	19
		4.4.2 Suggestion box	21
	4.5	Template and create email section	23
		4.5.1 Templates	23
		4.5.2 Storage	24
		4.5.3 Event attachment	24

	4.6	Additional functionality	25
		4.6.1 Shortcut key	25
		4.6.2 Context menu	26
	4.7	User warning	26
	4.8	Example output	26
5	Eval	luation	28
	5.1	User testing	28
		5.1.1 Observed behaviour	29
		5.1.2 Feedback via survey	29
		5.1.3 Indirect feedback	29
		5.1.4 Future evaluation	30
	5.2	Requirement evaluation	30
6	Futu	ire work	32
	6.1	Options page	32
	6.2	Extending input	33
	6.3	User interface	33
	6.4	Improving user experience	33
	6.5	Locales	34
7	Con	clusions	35
	7.1	Project summary	35
	7.2	Skills and knowledge developed through the project	35
	7.3	Reflections	36
Re	feren	ices	37
A	Part	cicipants' information sheet	39
B	Part	cicipants' consent form	40
С	Sign	-Up Sheet	41
D	Pre-	test Questionnaire	46
F	Dost	tost Augstiannaira	10
1	rost-test Questionnaire 4		

Chapter 1

Introduction

"Write Right" is an extension for the Thunderbird email client that helps students write and format emails to teachers. In Britain, there is a certain email etiquette that is expected. When not followed, a student can appear rude. It also takes a student longer to write a well-mannered email. Write Right allows students to enter a minimal level of essential details before automatically generating a well-mannered, appropriately formatted email to their teacher. This is not just about saving time: this etiquette is rarely taught to non-native speakers. My sister who is currently teaching English to young professionals in the south of France tells me she spends a week dedicated to writing and formatting emails. This is to European students who are much more exposed to British culture than students from East Asia for example, still needing a week to learn email etiquette. Just as a student would feel unfairly treated to be examined on a topic that has not been taught, so they have the right to feel unfairly demonised by unknowingly not complying with an etiquette they do not even know exists.

1.1 The importance of writing appropriate emails

Email is one of the most common ways for students to interact with teachers and thus a major part of a student-teacher relationship. A significant study (Stephens, Houser, and Cowan, 2009) was conducted to see the impact of overly casual email messages with teachers and the findings showed message quality has a significant impact on an instructor's willingness to comply with a simple request for a face-to-face meeting.

The Scottish government sees email and the ability to write a good email as a key part of the curriculum (Education Scotland, 2017) and they set a benchmark for first-level technology use to be able to communicate and collaborate with others through means like email. First-level in Scotland is a set of goals across a range of the curriculum such as technology, maths and literacy that the majority of kids should be able to do by the time they reach Primary 4. I remember spending at least 5 lessons in a Scottish high school learning how to compose and write a polite and well-formed email. This experience is not global, with different cultures having different views towards the importance, use, and way an email should be formatted (Sheng, 2022). This means that students coming to university in the UK understandably unaware of British email etiquette can be disadvantaged to the detriment of their academic performance. Note this that is "British (culture)" not "English (language)" because one of the inspirations for this project was an American lecturer working at a UK university complaining that British etiquette necessitated scripting an overly polite "request" that was actually demand to have a book returned that was borrowed "for a week" **six months ago**.

Planning and composing emails with the expected politeness and etiquette can be difficult for EFL learners (Economidou-Kogetsidis, 2015). Attitudes towards email differ globally: some use emails just as a way of contacting someone as informally as texting or social media (Sheng, 2022). But elsewhere a polite preamble is expected in emails before the main point. A student may well be being polite in their own culture without realising it is "just not cricket."¹ The Write Right extension streamlines this process and should reduce or even remove any anxiety a student may have about creating a well-formatted email.

1.2 Difficulties of making Write Right

Write Right has two major ways of inputting details for the email, manually through a dialogue box or restricted text entry in the email body. Both but especially the latter necessitate natural language processing (NLP) techniques to extract the relevant details. NLP is a complex field trying to solve tasks that humans find easy but machines do not. Contemporary NLP involves the use of advanced algorithms and statistical models, often on a mind-boggling scale, to process human language in a less accurate way than people manage apparently effortlessly inside their own heads. While there are many powerful NLP libraries available in the Node.js² ecosystem, it may not be possible to use them directly in a Thunderbird extension for reasons including but not limited to:

- Thunderbird extensions are built using the XUL³ framework or the WebExtensions API, which have their own APIs and interfaces that are specific to the email client. These APIs may not be compatible with NLP libraries designed for Node.js or other web development frameworks.
- NLP libraries may have dependencies or require features that are not available in the Thunderbird environment. For example, NLP libraries relying on external data sources or server-side processing, which may not be accessible from within a client-side application like Thunderbird.
- Thunderbird extensions are designed to be lightweight and have minimal impact on system resources. Using complex NLP libraries within a Thunderbird extension could potentially slow down the application or cause it to crash, which could negatively impact the user experience. Also, in Thunderbird, heavyweight processing is just not cricket.

¹An English (as in the country) idiom for something considered outside "the rules" that everyone knows because cricket is famous for having the simplest rules of any sport in history... not.

²Node is a back-end (server-side) JavaScript system but — confusingly to experienced programmers new to Thunderbird — is used client-side in Thunderbird.

³An in-joke based on a corruption of a quote from the 1984 Ghostbusters film: "There is no Dana [a character] only Zuul [a god controlling Dana's body]" to become "There is no data, only XUL."

These reasons and many more make detail extraction from completely free-form text much harder. Given the time restrictions for this project, it was necessary and eminently sensible to create a parsing expression grammar that works with a flexible set of grammar rules to extract details from limited contexts in a lightweight way.

Another challenge of creating a tool to help students write emails is creating a user interface that is intuitive and easy to use: a tool that is hard to use is not popular, although some common commercial software defies this. Introducing a new method of sending emails means that the interface must be well-designed and noticeably speed up the process of creating and sending an email.

1.3 Project parameters and limitations

The project supervisor had the idea for this plugin initially from the story where an American professor was justifiably irritated by British email etiquette. The idea resurfaced when the supervisor heard colleagues complaining about foreign students sending rude emails and thus denying those students' requests for meetings — remember Stephens, Houser, and Cowan (2009) from section 1.1. This gave the tool the initial target audience of international students but actually could be a time-saving and potentially face-saving tool for anyone, even native speakers of English. Being a native speaker of a language does not automatically mean you are "good" at it or enjoy writing in it. Even if you do, the sheer volume of correspondence is a considerable time-sink.

The tool is specific to Thunderbird as requested by the project supervisor. The extension is designed to work with all versions of Thunderbird after 78.0.

Chapter 2

Background

The most critical part of an email system is the ability to send (and receive) emails. As this is taken care of by Thunderbird, then the two most critical parts of this project are handling text input — so the message is both correct and appropriate — and creating an intuitive user interface (UI) with a pleasant user experience (UX) — so the tool is notably less work than manually composing your own British etiquette-compliant message. This chapter explores different options for parsing the text and justifies the decisions made for the tools and methods used. It also highlights the technical decisions made that affect the user experience. Alternative systems and tools currently available are compared and discussed, highlighting their short-comings.

2.1 Parsing text

A detailed understanding of parsing text is not needed for understanding this project. However, a general background helps understand the design choices for this project.

2.1.1 Parse tool choices

For Write Right to handle text input there must be parsing to extract information to then insert into the email template. Thunderbird limits the natural language processing achievable within the project time-constraints due to the limited libraries available within the Thunderbird ecosystem.

Information extraction¹ is a hard task even with the tools available. The three considered for this project were (in order of increasing complexity) regular expressions (REGEX), parsing expression grammar (PEG), and named entity recognition (NER). Creating a custom named entity recogniser able to perfectly identify details from the limited input is hard, even when the project supervisor is a successful domain expert. Moreover, NER is overkill as we can instead implement simple grammar rules for the user to follow which not only reduces complexity but is appropriate for the nature of the user task.

¹There is a field of NLP called Information Extraction (IE). This project is not IE (in capitals) but the core task is the same: extricate specific words from free-form text and use them to populate a template.

Thus we can use either REGEX or PEG to extract the details from these rules. PEG is a superset of REGEX and it seemed likely even from the earliest days of this project that REGEX's context-free nature lacks sufficient expressive power to do the task properly, thus ruling it out. Parsing expression grammars handle potential ambiguity better than REGEX, so if there is a conflict where a detail could be between options PEG will always have the same determined result (Ford, 2004).

Since the input uses rules to identify the details, it is important the user is made aware of working within the grammar's constraints. Superior error reporting for this is another benefit of using PEG over REGEX. Other benefits of using PEG include:

- **1. Simplicity:** PEGs are designed to be easy to use and understand.
- **2. Predictability:** PEGs are deterministic, meaning the parser always returns the same result for the same input, thus facilitating testing and debugging.
- 3. Error Reporting: PEGs have good error reporting capabilities.
- 4. Flexibility: PEGs are flexible and can be used to parse a wide range of input.
- 5. Integration: PEGs can be easily integrated into other software.

REGEX lacks these benefits; NER has them but can be altogether more complex. Thus a parsing expression grammar was the "winning" method. Aspects of the other two options still apply as PEG uses REGEX and the constructed system is a simple yet effective rule-based named entity recogniser which can correctly identify the location, time, and topic of a proposed meeting from text.

2.1.2 PEG.js

The tool PEG.js (Majda and Ryuu, 2010) was selected from a short-list to integrate the grammar into the extension. PEG.js allows creating a custom grammar built on rules using regex expressions which can be exported as a fully functional parser to a javascript module which in turn becomes part of the plugin. The parser provides a function that accepts a limited text input and returns a JSON (JavaScript Object Notation) data structure, which is ideal as the Thunderbird ecosystem is Javascript.

Since the Javascript runs on the client side, it cannot use libraries that would run using NodeJS such as NLP.js. This is the main limiting factor in making the decision for the parser to use, as there are limited options that work in plain Javascript. Options such as NLTK (Bird, Klein, and Loper, 2009) or Standford CoreNLP (Manning et al., 2014) are written in either Python or Java, thus for these to be used an interpreter would have to be implemented as well. This adds a substantial overhead outweighing any benefit.

PEG.js is a flexible tool that allows complex grammar to be produced and integrated into vanilla Javascript that can run client side: an essential requirement for Thunderbird. PEG.js can handle large inputs with minimal overhead, although not needed here. After extensive research, The options found that could be exported to Javascript were, alphabetically:

 All of the above complete a similar task with similar results, so any would suffice. All the options were given a fair shot by using each to write a small grammar and import it into the Thunderbird extension. They all work but the one with the easiest setup was PEG.js. Moreover, PEG.js had the best online playground which was the deciding factor. Being able to change values and explore the software easily before committing was important as parsing is something I had limited experience with at the time.

PEG.js uses a backtracking algorithm to match the input against the grammar rules. This means that it tries to match the input against all possible paths until it finds a successful match. Backtracking can be slow for certain grammars but PEG.js provides several options to optimize the generated parser's performance, one of which is memoization to cache partial results which massively improves the parser's overall performance when retrying parsing paths. If you already know a fragment is the meeting time, you do not need to keep reparsing it.

Disadvantages of PEG.js are limited community support and dreadful documentation. It is so awful it almost made me change tools. It exacerbates an already steep learning curve with undue trial and error because the provided examples give little detail to how to create a text parser. This is in sharp contrast to tools such as Chevrotain where there is a much greater supply of examples. Since many of the grammar rules were transferable, I was able to learn from external examples and bring it over to PEG.js. Perhaps this is why no one has bothered to improve what passes as documentation for PEG.js.

2.2 Technical background

2.2.1 Thunderbird

Thunderbird, a free and open-source email client from the Mozilla Foundation, is a popular alternative to commercial email clients like Microsoft Outlook and Apple Mail. Thunderbird's open-source nature and active development community make Thunderbird a reliable, flexible, and extensible email client for both personal and professional use. Thunderbird uses ES6 (ECMAScript 2015) in its codebase. This allows for new features in the Javascript language specifications. Extensions comprise a mixture of HTML/XUL, Javascript, CSS, WebExtension and Thunderbird APIs (figure 2.1). For a tool to run and have access to features like composing an email, the user must explicitly grant permission for the requirements defined in an extension's manifest document. The permissions used for Write Right's functionality and user experience are, alphabetically:

w.

XUL files typically have a cleaner separation between the user interface and application logic as XUL uses a component-based structure to separate functionality. XUL also



Figure 2.1: Thunderbird example of how the different files and systems interact (*Thun-derbird*, n.d.)

Original Text:

Sally I am starts work at yours monday from dave

Business version:

Dear Sally,

I hope this email finds you well. I am writing to let you know that I will be starting work with you on Monday. I am really looking forward to getting started.

If you have any questions or need any help with anything, please don't hesitate to get in touch.

Best wishes, Dave

Figure 2.2: Tweet showing the inputted text and outputted email from user @DannyRichman (Richman, 2022)

allows developers to create custom UI elements and modify them more easily than is possible in HTML. No prior experience in developing in XUL was not a problem, but when moving to the window approach instead of modal due to section 3.2.3 it was easier to develop in HTML compared to XUL.

2.3 Alternatives

2.3.1 ChatGPT

Early 2022 saw a substantial step forward in consumer-facing AI when ChatGPT (Chat Generative Pre-trained Transformer) became the fastest-growing consumer application ever (Milmo, 2023). ChatGPT is a powerful tool that takes prompt text and can create output in different formats including essays, poetry, film scripts, program code, and emails. A pertinent use of ChatGPT was demonstrated by a Twitter user who claims to have integrated ChatGPT with gmail to help someone with dyslexia write emails with professional etiquette from limited input, figure 2.2.

As powerful as such AI is, its output is non-deterministic which is problematic if it generates inappropriate text that the email sender does not notice or cannot understand. The AI can also "assume" details to be included ranging from harmless to harmful because it is merely generating the most probable text based on its training. listing 1 demonstrates this beautifully: the prompt was Write a formal email to organise a meeting to discuss natural language processing, on Thursday at 10:00 using teams. Despite the limited input, the AI's output is 1 useful paragraph written inappropriately followed by 3 paragraphs of ill-advised waffle. It talks of "bringing together a group of experts" when in reality it is one expert (the supervisor) and one non-expert (the student) and thus is wrong. The agenda is completely inappropriate because it is the recipient (the supervisor) who will likely steer the meeting. Even if the entire email were trimmed to just the first paragraph, an academic already intolerant of "ill-mannered" emails is not going to be pleased to read about being "formally invited to a meeting" in their own office by a student of all people. The content of this AI-generated email may well trigger a more hostile reception than the typical student email, exacerbated by the undue length adding or possibly multiplying the aggravation factor.

One of the main benefits of using Chat GPT is that it can handle a wide range of natural language queries and produce responses that are often indistinguishable from those of a human and thus may have smoother output than Wite Right. However Write Right reliably and consistently does only as it is told using only the details provided.

2.3.2 Other Thunderbird extensions

Thunderbird has a huge community that continues to develop and create extensions. Some of the extensions perform similar tasks to Write Right, but none that I could find to the best of my ability were able to take in a small text input and extract the details to a templated email. Dear [Recipient],

- I hope this email finds you well. I am writing to formally invite you to a meeting to discuss natural language processing, which is scheduled to take place on Thursday, at 10:00 AM. We will be using Microsoft Teams to conduct the meeting.
- The purpose of this meeting is to bring together a group of experts to discuss the current trends, challenges, and opportunities in natural language processing. I believe that your expertise in this field would be valuable to this discussion, and I would greatly appreciate your participation.
- The meeting will last for approximately one hour, and we will be covering topics such as machine learning algorithms, speech recognition, and semantic analysis. The agenda for the meeting will be circulated prior to the meeting.
- Please let me know if you are able to attend this meeting by responding to this email. If you are unable to attend, please let me know and I will arrange an alternative time.
- I look forward to hearing from you soon.

Best regards,

[Your Name]

Listing 1: ChatGPT email generated from meeting prompt

SmartTemplates This extension has a similar idea to Write Right, albeit entirely independently. It allows the user to create and use email templates with variables in the templates. The user is able to edit the templates or the variables in the extension. This extension has the drawback of the user having to enter the value for the variables to be replaced individually: there is no way to turn a short part of the text into the template. The tool always includes the full template in the email, so it does not adapt to still make sense when details are omitted. The tool is not that intuitive for a novice user; template creation can be complicated, with no real help to the user on what details they can include or how to include them. This extension requires the student to write a well-formatted email template in the first place which is self-defeating in our scenario. It is also a potentially daunting task, which is why my tool comes pre-loaded with default templates. A final drawback of SmartTemplates is that it requires a license to use the tool after the trial period. Write Right should be open source and free to the user no matter how long they use the tool.

QuickText Similar to SmartTemplates, Quicktext allows you to create templates for emails and quickly insert these into your compose message. This has similar drawbacks to the SmartTemplates extension where the user is still required to create the first template. It does not handle any detail extraction and insertion into the template either. The user interface for this tool is much more intuitive than SmartTemplates's and has helped inspire design decisions made when implementing Write Right's user interface.

Other Extensions The two extensions above are the closest extensions I could find to the desired functionality of the Write Right tool. However, there are other tools that still help speed up writing emails such as StandardTexter which can store pre-made text and then retrieve and insert them during email composition. None of the extensions I could find have a user interface where a user can insert details either manually or by text and then produce a template. Specifically, there are no extensions I could find that are designed to help students write "proper" emails.

Chapter 3

Design

3.1 System design

3.1.1 Design method

There was a preliminary design phase prior to the commencement of implementation. Where rough ideas of functionality that I believed would be important for the final tool were mapped out. Starting with a rough preliminary sketch, gave me a starting point to break down my development into sections. When sketching out the tool I used Figma (*The Collaborative Interface Design Tool.*). This is a design tool that allows the designer to create and reuse components and create interactive prototypes. The user interface design can be developed gradually as features and implemented. Starting with a rough preliminary design was also beneficial as it meant I had not spent a huge time on the first design when functionality was added and dropped throughout development I could go back and add to the design using an iterative approach.

This approach allowed me to have a very basic but working prototype early on in the development process. This was useful as it meant throughout the year there was a most recent working version of the system. It allowed for protection against unforeseen setbacks, such as the bug discussed in Section 3.2.3, where development was cut short and some time was spent organizing meetings and trying to find a solution, there was still a working system to experiment and test on. This approach also left me to write code that can be moved around and changed easily. For example, after some informal user testing of a mock-up of the user interface, I decided to change the design from a section drop-down to a layout that shows all the options to the user at once. Having not been committed to a certain design from the beginning allowed me to try out many designs to see which work better than others.

This does not suggest that a formal design stage is not a valuable method. Having a strong concept design from the beginning would've allowed me to really think about aspects of the system that were most important. It would also have allowed me to have less tech debt to deal with. As for each different version I developed to get the user's opinion often garbage code would be left behind or re-implemented leading to poor code being in the system that was required to be cleaned up. Overall, this approach to design suited the requirements for this project well.

3.1.2 System requirements

As with the design process, the requirements capture was also informal. The requirements and priorities of the tool shifted over time based on possibilities with Thunderbird, user testing, and discussions with my supervisor.

Usability To ensure that the tool is useful and accessible to all students, especially international students (see Section 1.3), it's crucial that it allows for the creation of well-formed and polite emails with ease and speed. The tool should be designed to assist students in crafting emails that adhere to the conventions of British email etiquette, while also streamlining the process to save time and effort. By prioritizing ease of use and accessibility, we can allow the majority of students to create well-mannered emails, regardless of their background or experience.

User Experience It is important that students do not find the tool excessively tedious to use - if most would prefer to just write a long email themselves or replace the values from their own templates themselves the extension is unlikely to be used. The user should have different options and know what their options are.

Compatibility Since it is a requirement for this tool to work on Thunderbird from Professor Mitchell, the plugin should be available across all different operating systems. Since Thunderbird has not officially released a phone version application for their software I did not have to worry about this. For certain WebExtensions to work as intended I had to use a minimum version of Thunderbird 78.0.

Maintainability In order for this tool to be useful and successfully developed after release, it's crucial that it can be easily modified and maintained by other developers. To achieve this, the system should allow for the addition of templates without altering the underlying code. Moreover, new grammar rules should be able to be integrated into the parser with minimal changes to the codebase. To ensure that future developers can make changes confidently without breaking major functionality, a comprehensive testing suite should be implemented. By prioritizing ease of modification and maintenance, we can ensure the longevity and usefulness of this tool for years to come.

Security Security is not a critical concern for the requirement gathering as there is limited sensitive data being collected, with the data that are collected explicitly permitted by the user. Since the tool runs on the client side of the users' system, there are limited risks. Since some modules of the tool have access to certain WebExtension API's it is important to avoid the possibility of cross-site scripting (XSS) attacks, which could come from the text-input subsection 4.4.1 or from the custom template upload subsection 4.5.1, but once again since this is only running on the client side the only data that is accessible to them would be data that is already available to the user.

Accessibility To ensure the system is accessible to a wide range of users, it should be designed with different needs in mind. For instance, users whose first language is not English should be able to use the system comfortably, and to facilitate this, options

Extension Name	Date	Users
Provider for Google Calendar	01-09-2021	242,202
Manually sort folders	31-08-2021	205,265
ImportExportTools NG	09-12-2020	201,085
LookOut (fix version)	30-08-2021	157,337
Quicktext	16-08-2021	123,930

Table 3.1: Table showing the top 5 most-installed Thunderbird extension. Information obtained on the 25/03/2023 from *Thunderbird addon code knowledge base*

beyond text entry should be provided for creating emails. Additionally, users with dyslexia or other reading/writing difficulties should be able to use the system with ease. To accomplish this, both text-to-speech and speech-to-text capabilities could be implemented. By prioritizing accessibility and inclusivity, we can ensure that all users can utilize the system effectively and without barriers.

3.2 User interface

This section describes pertinent design decisions made and how those decisions were affected by feedback from user testing. The user interface of the tool was designed to be easy for users to quickly familiarise themselves with. It should make the process of writing a well-formatted and written email quicker than having to write it out themselves. The design of the interface was altered and updated throughout the process, to either work around bugs or due to feedback from user testing. It was important to continue to use an iterative approach as often the way a designer plans for a system to be used and the way a user tries to use the system can be very different things.

3.2.1 Extension layouts

Early in the design process, I decided a form-based approach would be the most user obvious and friendly. Then reading into some of the Thunderbird documentation I saw that they incorporate pop-up functionality. Both of these reasons led me to choose a pop-up form-type design for the extension.

To get inspiration from what is popular and possible with a Thunderbird extension, I looked at the 10 most popular extensions for Thunderbird, Table 3.1. Most of the top 5 extensions are fixing or doing work within Thunderbird to streamline the application or provide missing functionality that other email services offer. Quicktext is a text generator that quickly adds a signature to the email. Parts of this functionality provided inspiration and were added to the context menu so that users can quickly add some details to their email without having to use the full tool.

The first tool with a pop-up type interface that I was looking for was the SendLater extension (Jonathan Perry-Houts, 2023), it is the 9th most installed extension on Thunderbird. This was my main inspiration because this is the most popular extension to make use of a popup window, along with multiple options for inputting the values without overwhelming the user. As my project overview diagram shows Figure 1, there are two ways for the user to input the details for the emails; manually, or by text entry. SendLater extension does the same as it has an input box for limited grammar or manual input where you can select specific values.

The initial design included a collapsible menu, where you could drop down to see the manual entry details or the text entry details, however, after carrying out some informal user tests among my peers I realized that this ended up being more confusing despite hiding fields. Instead, I went for a more simple design with everything being available to the user at once, but each section separated so a user knows they only must do one.

3.2.2 Overall aesthetic

I made the decision to go for a simple visual style, with no animations, and a default grey-scale colour design (however in future work I would like to extend the color design to work with custom themes for the email client). The simple design allows the user to see all functionality available to them from the point of contact. Since the tool is being used as an alternative to writing emails in the normal sense, then this tool should be designed in a way that it is easy for the user to understand what is needed from them and should speed up their process of email writing rather than hinder it. A simple design is also much easier to implement as I have little experience designing an aesthetically pleasing interface, it allowed me to focus more on the importance of working on functionality rather than extra design touches.

The overall design will end up being similar to a form design with different types of fields for the user to fill out. Popup forms have become a data-gathering, marketing tool most websites employ now. This can lead to users being discouraged when they see a popup form. This negative feeling sometimes occurs from users being frustrated about how many fields they have to fill out (*Popup forms: Bad practice or best practice?*, 2022). That's why the Write Right tool form will work and create an email as long as at least one field is filled out. This design should be intuitive for users to understand what is needed from them to fill out the details for the email as forms are something most users will have filled out before.

When creating the form I made sure to have either labels or placeholder text to represent to the user what the field was expecting. Through the clever placement of labels, instructions to fill in a field and additional design elements can make a form less daunting and may result in fewer mistakes made (Jarrett and Gaffney, 2010).

3.2.3 Problems with modal dialogs

The manifest.json file has a *default_popup* on the *compose_action* setting which can be used to create a popup modal on the compose window. However, there has been a bug since 2016 (*1292701 - autoclose popups shouldn't close when they open a modal dialog* (*e.g., file picker*)) where opening up the file picker modal would close the popup window removing any information in this. This became an issue when adding in the custom template functionality Section 4.5.1, where access to the file explorer was needed but the popup window also needed to remain open so the user can upload the template.

To overcome this bug I organized a meeting with some other Thunderbird extension developers and developers who contribute to the open-source code of Thunderbird. In this meeting, the developers agreed with me that this bug is related to the issue I was having and they suggested that I allow the users to upload templates through the options page, or change the popup modal to a popup window. I also came up with an alternative idea of allowing the user to write the template in the compose message body, and then use the context menu or a shortcut to turn the selected text into a custom template. After raising the issue with the developers it has since been promoted in priority and has been assigned to a developer to fix. So hopefully, this won't be an issue for future developers.

I ultimately decided to change the popup modal to a window. I think this is the most obvious way for a user to upload their own custom templates, and this belief is backed up by the user testing carried out in section 5.1. As with a choice in the context menu or options page, this functionality may get lost as a new user of the tool may never notice these aspects. Along with making the user aware of errors missing in their custom templates is done much easier in the popup window as composed to the context menu and options page Section 4.7.

3.3 System architecture

From the outset of this extension's development, my intention has always been to release it as open-source software, allowing the community to contribute and enhance the tool's functionality. To facilitate this collaborative process, it was crucial to ensure that the codebase was well-structured and thoroughly documented. By taking these steps, other contributors would be able to easily navigate the code and understand its functionality, making it possible to improve and refine the tool over time.

The software architecture for this tool is based on the manifest.json file which serves as the entry point for the extension. It also gives access to the permissions, and resources mentioned in Section 2.2.1. This manifest file links with the background JavaScript file which handles the compose action, opening the popup window, and the shortcut keys to create a template. The popup design and layout is handled by the popup.html and popup.css files. The functionality is handled by different Javascript modules, alphabetically:

 createEmail 	 fileSaver 	 popup 	 templates
 createEvent 	• parser	 suggestionBox 	

Each module is responsible for its own functionality allowing for a more modular and maintainable code base. The different modules interact with only the modules required and are only given the information they need to run successfully, see Figure 3.1.

In addition to these different modules linking together, this tool interacts with WebExtension APIs from both Thunderbird and Mozilla. These extensions provide additional functionality such as composing an email and getting details from a user account.

Overall, this architecture allows for a clean separation of concerns and facilitates the development and testing of individual components.



Figure 3.1: Module Architecture Layout

Chapter 4

Implementation

4.1 Implementation method

This project was implemented end-to-end (Thomas and Hunt, 2019, Tip 68). Within 4 weeks I had a basic but complete prototype that took and parsed a structured sentence from a simple pop-up dialogue to generate an appropriately populated template email. With proof-of-concept complete, this left all remaining development time for evolution. The more I read the Thunderbird documentation, the more seemed possible. I followed most of the agile development principles, especially feature-driven development (Tirumala, Ali, and Babu, 2016), combined with tracer-bullet development (Xu, 2020), a metaphor for software development based on instant, real-time feedback under actual conditions. Its purpose is not to write prototype code that is discarded but clean working and feedback compared with a traditional waterfall approach (Thesing, Feldmann, and Burchardt, 2021) which requires more due diligence to be carried out before hand. This rapidity suits a student project timescale.

4.2 Contact detail section

Contact details form the upper quarter of the user interface, Figure 4.1. There are text input fields for Recipient Email, Recipient Full Name, and Sender Full Name, along with a drop-down for the recipient title. The tool is designed to be modular so if details are missing from some sections it should not stop email generation as long as one of the middle sections contains details. The contact section is not vital and an email will still be produced even if all the details in this section are omitted, though in that case it will be missing the names at the bookends of the email, and the To field will be blank.

To assist the user and streamline completing these details, the recipient email field utilizes the contacts API endpoint to allow auto-completion of the input field for the contact email. If a user omits these fields on the plugin then the automatic generation of an event invitation (see Section 4.5.3) cannot happen. The user is warned of this.

k.

Emailify - Mozilla Thunderbird -	
Recipient Email]
Recipient Full Name	Title v
Sender Full Name]
Manual Entry	
Select Type of Email	
Text Entry	
Meeting to discuss Natural Language Processing Course, using Microsoft Teams on Thursday at 14:30.//	
Suggestion box:	
Add 'Meeting' or 'Question'	
Template Selection	
Email Template	
OR	
Browse No file selected. Template name	Upload
Create Email	

Figure 4.1: Default User Interface when opened

Meeting	v
Meeting topic	
Meeting Location	•
Date for meeting dd / mm / yyyy	
Time for meeting :	
(a) Meeting email manual entry	
Question about something in class/coursework	~
Course name	
Question Topic	
Specific part of question	
(b) Question email manual entry	

Figure 4.2: Different kinds of manual inputs

4.3 Manual entry section

The next section of the user interace (Figure 4.1) handles manual gathering of details. For this plugin to be accessible by more people, there must be a manual entry option in case users prefer this to using natural text or were confused by the grammar rules they must follow. Write Right has been designed so that there are no details you can enter in the manual entry that can't be included in the text entry and the same is true vice-versa. It also takes the ambiguity out of it for the user as it is clear what each entry is used for and what is missing (a user will still be made aware of missing details whether using manual or text entry Section 4.7).

There are two different types of manual entry forms to fill out depending on the type of email the user is trying to send. Figure 4.2 demonstrates the two different kinds of email forms that can be available to the user; one that includes different input formats such as date and time along with an open text box to put the meeting details down, the other handles the question form which just consists of three text fields: for the course name, question topic and specific parts of the question.

4.4 Text entry section

4.4.1 Parsing

To be able to allow flexible enough grammar that details can be included but also not so wide that it becomes too hard a task to parse, I decided to set up the input text to be parsed when certain triggers are hit, for example, most words are ignored and labeled as "connective" unless they are preceded by "to discuss" or "about" in this case the words are labeled as Topic Words until the user places a ",", or they are a hard-coded entity such as "teams" and "zoom" for the meeting location. This format allows for any length of the text to be entered as the topic sentence for the meeting or question.



An example of what a user may enter is the following *Meeting to discuss natural language processing, using teams on thursday at 10:00.* This creates the syntax tree shown in Figure 4.3a. Whereas for a question type of email, the user could enter the following *Question for CS109: What is the point of a list? Specifically when would I use it instead of an array?* which creates the syntax tree shown in Figure 4.3b. As is clear the different types of emails expect different details to be included. The details the parser is expecting depends on the first keyword used either "Meeting" or "Question". The parsing is handled by inputting the text into the parse function produced by the PEG.js module Section 2.1.2.

When using a parsing expression grammar the choice operator chooses the first match in PEG whereas in a context-free-grammar (CFG) it is ambiguous (Mascarenhas, Medeiros, and Ierusalimschy, 2014). The list for grammar to check goes like this:

- 1. Subject ("Question"/"Meeting")
- 2. Space (" ")
- 3. Question
 - (a) Course ("for" + course_name + ":")
 - (b) Topic (":" + topic_words + "?")
 - (c) Specific topic ("specifically" specific_topic + "?")
 [nosep]
- 4. Location ("Appleton"/"Office"/"Teams"/"Zoom")
- Day ("Monday"/"Tuesday"/"Wednesday"/"Thursday"/"Friday"/"Saturday"/"Sunday"/ddmm-yyyy)

- 6. Topic ("to discuss"/"about" + topic_words ",")
- 7. Time (00:00)
- 8. OptionalTime ("when works best for you"/"what time suits you"/"I can work around your schedule")
- 9. Connective (Other words)
- 10. eof (End of file)

This allows a deterministic result for creating the two different types of email. To break down what some of these tags mean, *Subject* refers to whether the text is a question or meeting, *Question* refers to a different branch where only the course, topic, and specific topic can be included in the email as the rest of the tags are not necessary for a question email, *Connective* is just a regex expression to represent other words/numbers/punctuation that could be included in the text so the parser does not return an error when characters that are not expected are included.

4.4.2 Suggestion box

When developing the text input functionality and running some informal tests with peers I noticed that some of the rules could be forgettable when entering text. This led to users forgetting what details they can include or how to format their questions or meeting topics. I believed that the grammar was simple enough that once learned it would become a much easier tool to use so instead of changing the grammar I decided to add a suggestion box.

The suggestion box goes underneath the text-entry field and reminds the user of what details they are missing and also what rule to follow. The suggestions are only those that work with the current inputted text. For example, if nothing is typed in it will show a suggestion to start the input by putting either *Meeting* or *Question*. Once the extension knows the type of email you are going to send (question or meeting) it will then show you only the applicable rules. Figure 4.4a shows you can add any of the details that are specific to requesting a meeting. For a question email type, the suggestion shown would be "Add 'for' then the course name" since that is the only applied rule. This is the same case in Figure 4.4b where after to discuss tells the user they must enter a comma to end the topic sentence before adding more details.

To improve ease for the user they can click any of the suggestions and it will add the suggestion text into the field. Chapter 6 discusses the idea of extending this functionality to enable hotkeys so that the user does not need to switch between keyboard and mouse. The hard part of implementing this was making sure that only the rule that could be applied was shown, and that crossover rules from different types of emails were not suggested. For example, if a user is trying to organise a meeting but it is suggesting that you add a specific question this would be confusing for the user and wrong.

Meeting	1.
Add day or date Add time or optional time (what time suits you) Add location (appleton/office/teams/zoom) Add 'to discuss' or 'about' (a) Possible next entry details for a meeting subject	
Meeting to discuss	

Add a comma after meeting topic

L

(b) If a certain rule must apply (such as a meeting topic has to end with a comma) only that suggestion is shown.

Figure 4.4: Two suggestion box examples at different stages in the text input process.

4.5 Template and create email section

4.5.1 Templates

The implementation templates were split into two categories: default and custom. Default is the templates provided by the extension, which are: formal, friendly, and urgent. Whereas custom templates are those uploaded by a user. For all templates, there are pointers to where details will be inserted when the text or manual input is parsed.

Meeting Template

• [name]	• [day]	• [meeting_location]
• [meeting_topic]	• [time]	• [sender_name]

Question Template

• [name]	 [question_topic] 	• [sender_name]
• [course]	 [specific_topic] 	

Default Templates These are stored in a Javascript file that contains two objects to represent the different kinds of emails (meeting and question). The template is broken into sections (Listing 2) so that the template can be malleable and make sense and be grammatically correct if some details are omitted.

```
const meetingTemplate = {
1
      "friendly": {
2
           "intro": "Hi [name]! < br>< br>I hope you're doing well
3
               !",
           "meetingtopic": " I wanted to reach out to you about [
4
              meeting_topic] and see if we could schedule a
              meeting to chat. I have a few questions and
              concerns that I'd love to discuss with you.<br><br>
               Are you available to meet",
           "day": " on [day]",
5
           "time": " at [time]",
6
           "onlinelocation": " using [meeting_location]?",
7
           "offlinelocation": " at [meeting_location]?",
8
           "rest": " If that doesn't work, let me know what other
9
                days and times might work for you. <br><br>> Thank
              you so much for your time and help! Looking forward
                to catching up soon. <br><br> Best, <br>[
               sender_name]"
       },
10
       . . .
11
12 }
        Listing 2: Snippet from template is showing the meeting template object
```

If no template is selected then friendly will be chosen by default. The default email template to be chosen is something I would like the user to select when an options page is implemented in the extension (see Chapter 6).

Custom Templates To create the custom templates the users can upload a plain text file from their system to be used as a template, they can name their template and then it will be stored and available like the rest of the templates. The user guide for the extension, explains how to go about creating a template and what must be included to make it work with the extension. A user will be allowed to upload a template with missing tags, although there will be a warning message Section 4.7 telling them of the missing details. To delete a custom template the user has the option to delete a custom template, the user interface for this is not the most intuitive at the moment due to the limit on time for implementation and the bug discussed in Section 3.2.3 delaying this part of development. A user can delete a custom template by selecting their custom template and then pressing the remove from storage link under the template selection. This can only be done with custom templates. In Chapter 6 I talk about plans to move this functionality to the options page, where a user can remove multiple templates at once. I also plan to extend the functionality to warn the user that they can't upload any more templates due to hitting the default storage limit (see Section 4.5.2).

4.5.2 Storage

To store templates in Thunderbird I had a choice between the CloudFile Thunderbird Webkit API extension or the storage web extension provided by Firefox. The latter stores the data on the local user's computer allowing up to 4Mb to be stored, and for the data to be stored through sessions and to be cleared when the user deletes or clears the extension. The user data is stored with a key value pairing, this allows multiple saved items to be returned quickly. On the other hand, CloudFile allows the linking of a cloud server to the extension so that templates can be uploaded and stored to the cloud. The benefits of this being more templates can be stored, and it would be easier to share templates with another user, the drawbacks of using CloudFile is that a user would need access to the internet to gain access to custom templates, along with more security checks to be implemented for uploading and retrieving templates from a cloud storage service. It could also require more setup for the user to connect to the cloud service storage. Finally, it requires more setup in development with more apparent possibilities for errors. For these reasons along with 4MB should be enough for the majority of users' custom templates I decided to use the local storage.

4.5.3 Event attachment

Since a major aspect of this tool is the ability to write emails to organize meetings, my supervisor suggested the extension should attach an invitation to a calendar event using an iCalendar file (ics). An ics file is made up of properties next to their values. Since certain properties and details are needed to make a well-formed event invitation it is important that I implemented some validation checking to make sure that the format of the file was at least correct and not malformed.

- 1. Organizer Email4. Start Time
- 2. Summary
- 3. Location

- Start Time
 Attendee Name
- 6. Attendee Email

The start time was the hardest part of this as it required combining two details inputted from the user and converting them to a different date and time format (yyyymmddThhmmss). This should not be a real issue except that the user can put in a day for the meeting date. This meant I had to create a day-to-date conversion function that took the system's current date and figured out the closest upcoming date with that day. This also inspired me to create the reverse function that checks if a date is within 7 days of the current date, if so convert it back into a day before inserting it into the email to make it a more readable email. To make the user aware of any details missing a popup will appear alerting the user to the missing details, the email will still be created to ensure that a user is never slowed down by the process however the event will not be attached.

4.6 Additional functionality

After the core functionality was implemented and working. I started to explore other ways of trying to make the tool more usable. I came to the conclusion that sometimes a user will not want to fill in the form to send an email, they might not even want to open up the form, remember *Popup forms: Bad practice or best practice?* I came up with two alternative options for using the tool without having to open the pop-up. The architecture and the implementation style I decided to follow made adding these additional features later on much easier than at the beginning. I implemented shortcut keys as well as context menu options. Both the shortcut and context menu, are things that are defined in the manifest and then their states are checked by the background script.

4.6.1 Shortcut key

This allows a user to type a short text such as:

Meeting to discuss natural language processing, using teams on thursday at 10:00

into the compose section of an email and then use a predefined shortcut key to turn the text into the default template. Currently, there are a few drawbacks for the user if they are using this method:

- 1. The user can't create an event attachment
- 2. The user can't change the default template
- 3. The user can't change the shortcut keys

Chapter 6 discusses solutions for these except the first. This was a deliberate decision since the event attachment will not have enough details without using the popup window. Along with a malformed event attachment being a user error that does not complete intended functionality (see Section 4.7), the user should be made aware of why the calendar attachment is not there in the event of a poorly formed attachment.



Figure 4.5: Context menu example of converting an email to a template

4.6.2 Context menu

Another option for the user to create an email from a template is by accessing it from the context menu. To implement this I used the MailExtension API library "menus". The menu only shares issue 1 with the shortcut approach, as once again, not enough information is given for a complete attachment to be created. As Figure 4.5 demonstrates, the user is given all default templates as options. Chapter 6 talks about future options to expand the context menu options to make it more useful.

4.7 User warning

I decided to use two types of errors for this tool: one that just warns the user of the error but will still complete the intended functionality of the user, and the other that warns the user of the error but also does not allow the intended functionality to happen.

The first type of error could appear when a user uploads a custom template that is missing one of the key tags for the template or when they try to create an email but are missing details from either the manual or text entry box. The reason I chose to just warn the user of these errors but still complete the functionality they are trying to complete as a user may not want to include a [time] tag for example in their templates, so they should be told that it is missing in-case they have just forgotten to include it, but should still be able to be used if they had missed out the detail on purpose. This error appears in the popup just above the create email button. Figure 4.6 shows the error message displayed when the time is missed from either the text or manual entry.

The other kind of error is when the functionality is still not carried out if the user misses details. This error occurs when the user tries to attach an iCalendar event but they are missing one of the key details mentioned in Section 4.5.3. Instead of just displaying the error message in the pop-up which could be ignored instead an alert box is opened warning the user they are missing details Figure 4.6 and in this case, the attachment will not be included but the email will still be created.

4.8 Example output

Listing 3 shows an example friendly email created by Write Right from the text *Meeting* to discuss natural language processing, on thursday at 10:00 using teams. The same email with manual entry would be produced by entering the same details. This example does not have the same issue that Section 2.3.1 discussed of including unwarranted details. This tool is also more beneficial than the other extensions that are currently

available and discussed in Section 2.3 because if you changed the input prompt to be the same minus the time. The second paragraph from Listing 3 will change to begin with "Are you available to meet on Thursday using teams?" This shows the benefit of the malleable structure of template creation of the Write Right tool compared to other template based tools currently available. Overall this example perfectly shows why a tool like Write Right will be useful as it only includes the details you want it to include, no more, no less, while still producing a polite a well-formed email.



(b) Error message to the user of event attachment details missing

Figure 4.6: Different types of user warnings

```
Hi John!
```

- I hope you're doing well! I wanted to reach out to you about natural language processing and see if we could schedule a meeting to chat. I have a few questions and concerns that I 'd love to discuss with you.
- Are you available to meet on Thursday at 10:00 using teams? If that doesn't work, let me know what other days and times might work for you.
- Thank you so much for your time and help! Looking forward to catching up soon.

Best, Adam Smith

Listing 3: Write Right email generated from meeting prompt

Chapter 5

Evaluation

5.1 User testing

To understand interaction with the tool I conducted a small scale (n=5) but authentic user test on a group of students from the University of Edinburgh. This was to identify usability problems and improvements to the functionality or the layout of the design. The testing process involved recruiting a group of participants who matched and spanned the tool's target audience. The 5 students consisted of two international students not with English as their first language (L1 in linguistic terminology), one international student with English as their L1, and two further non-international students with English as their L1.

Participants completed a pre-testing survey. The response help check if our testing is relevant to the target audience. It also identifies potential users' motivation for using the tool by gathering information such as how regularly they use email, what kind of emails they typically send, and crucially whether they have ever been nervous about composing an email.

Participants were all asked to use the tool to carry out a set of tasks designed to mimic real-world scenarios and to test different aspects of the tool's functionality. Tasks had to be completed in the following order:

- 1. Create a question and meeting email using manual entry
- 2. Create a question and meeting email using text entry
- 3. Create a meeting email that includes the iCalendar attachment
- 4. Create an email using either a shortcut or context menu
- 5. Upload and use your own custom template

During each task, each user was encouraged to talk about what they were expecting and what they were doing — cognitive walkthrough (Salazar, 2022a,b) — so I could note any pain points. I would give limited help for completing a task unless it was a problem the user was having with using Thunderbird itself which is not what we are testing. If they could not complete a task due to the extension's shortcomings they were told to move on. Once they had completed or attempted all tasks they were asked to complete a debrief survey.

5.1.1 Observed behaviour

The users were asked to see if they could create an email with both manual and text entries. When the users were first shown the extension and were given a brief overview of how it should work. None of the users struggled with creating emails using manual entry. With the text part of the section, one user tried to start the text input with something other than 'Meeting' or 'Question'. Once this was corrected all users were able to create an email using the text entry.

All participants were able to upload a custom-made template, name the template and choose it to be used for the email. This reinforced the decision made in Section 3.2.3, as other methods of uploading files may have been more hidden and not been so useful.

Participants were aware of what each section was for and were able to fill in the details without needing to be prompted. This demonstrates the form-type page was a successful way of gathering user details. One participant was confused by the two different kinds of entries between the manual and the text but they checked the user manual and understood they must only fill in one of the two.

5.1.2 Feedback via survey

No particularly useful insights were gained from the use of the survey. The design of the survey was flawed in that it was trying seeking to evaluate the extension generally, rather than answer some particular question about the user experience. With a better-designed, more focused survey it would've been possible to gain more insight into a particular question to do with user experience. Along with the small sample size making the survey results we did get were not that reliable as they don't represent a group of users.

5.1.3 Indirect feedback

When using the text input the majority of participants commented on the usefulness of the suggestion box in reminding them what details they can include and how to include them. They commented how the grammar rules could've been more forgettable if it wasn't for this feature. One participant made a comment about how it was slightly off-putting that the Write Right theme did not match that of the Thunderbird client.

One participant commented on only being able to select from a limited list of possible locations for a meeting to be held in. When selecting a location from the manual entry it allows you to select others and input the location, this is not the same for text entry where it looks for a named entity. This is something that could be explored with more user testing, as if many users find this too limiting, then it could be changed to take a location in a similar way a meeting topic is (between "to discuss" and ",").

The two participants whose first language is not English both made a comment on how they enjoyed the context menu feature for quickly converting a short bit of text in the compose window to a fulfilled template. They also talked about how they would like this to be able to convert a small part of the email instead of rewriting the whole email, this is something I will discuss further in Chapter 6.

5.1.4 Future evaluation

User testing was a useful way to get feedback and observe the way a target user may interact with the tool. User testing usually is made up of a smaller sample size, this is beneficial to get quick feedback about the interface or spot any major pain points. However, the smaller sample size does mean that the feedback must not be taken as definitive, as this will not be representative of all target users, or potentially even the majority. In future evaluations, more effort could be done to get more users and a more varied representation of the target audience

Beyond user testing, a user study could be conducted. This would allow a deeper understanding of user preference and behavior. This could be carried out with a larger sample size, with the participants completing a survey after using the tool for a period of time. The goal being to gather data on user needs, motivations, and other pain points, which can then be used to inform the design and development of the tool.

To aid with the design and development it could also be useful to conduct more AlB testing, a small amount of informal AlB testing was carried out when comparing the user layouts discuss in Section 3.1.2, however, this was done with a very small sample size and was informal testing. To gather more reliable data from AlB testing it should be conducted on a large sample size of the target audience over a longer period of time so that a user's preference is not decided but what they are more used to.

5.2 Requirement evaluation

Another way of assessing the success of the tool is to compare it to the project requirements outlined above (Section 3.1.2). This comparison will also highlight areas that will require further work that will be discussed in Chapter 6

Usability The feedback from the participants provides an encouraging indication that students were generally able to use the extension to create an email through both input means. In the post-test survey, all participants answered *Yes* to whether the tool made email writing faster. Further usability improvements could be identified through further user testing, especially using AlB testing across a larger sample size.

User Experience The quality of user experience was more difficult to measure directly. When answering the survey at the end of completing the tasks all users commented on the ease of creating emails. Having a more in-depth post-test survey could give more information about user experience.

Compatibility The tool achieves the compatibility requirements set out. The extension was tested across the Linux, Mac, and Windows releases of Thunderbird, with all functionality working as expected. Along with the extension working with a minimum version of Thunderbird 78.0.

Chapter 5. Evaluation

Maintainability The system architecture allowed for a maintainable code base. Functionality was able to be added late on in the project due to this architecture. More work could be done to increase maintainability, such as not hard-coding values on the interface this would allow easier implementation for other locales. I could also have written more tests so other developers are able to add code and check existing functionality is not affected.

Security As mentioned when laying out the requirements for this tool security is not a major issue. This remained the case through implementation, although it was important to make sure the tool was only requesting access for permissions it needed and that inputted values were being verified.

Accessibility From the results above, all students were able to complete the major functionality of this tool by creating an email. This shows that the tool meets the accessibility requirement as students whose first language was not English were still able to use the tool. Missing functionality such as speech-to-text, text-to-speech, and other hotkeys could be implemented to further increase accessibility.

Chapter 6

Future work

The idea when creating this extension is that it will be developed to be open source at the end of the project. Therefore it was important to follow good design principles like Don't Repeat Yourself, splitting up functionality and keeping good documentation. This allows the codebase to be read by future developers so they could possibly add some of the functionality. Some of the possible functionality ideas will be discussed in this chapter along with the limitations that stopped them from being possible during the development of the extension.

6.1 Options page

An options page is a common occurrence with extensions. It allows the user to set default rules or preferences for the extension, or provide the extension with additional details, such as if we had chosen the cloud storage option in Section 4.5.2, we might need to get the user to give details to their own personal cloud storage. For this extension at its current stage, I see possible options that could be available to the user could be things like: allowing the user to choose the default template Section 4.5.1, whether iCalendar attachments should be attached automatically Section 4.5.3, change the default storage limit for Thunderbird extensions, delete or upload their own custom templates (via the options page instead of the popup), change the shortcut for creating a template from the compose message and finally eventually it would be good to allow the user to extend the grammar on the options page, however, this would require the user to have some knowledge in writing a parsing expression grammar so this is something that would have to be thought about carefully before implementing it. Time was the major factor that limited this from being implemented by the time I was aware an options page was available and had thought about the plan of it, I had to make the decision to dedicate more time towards fixing and improving the base functionality of the plugin.

6.2 Extending input

To extend the grammar to include more trigger words to flag details is not a hard task to do once you understand the parsing expression grammar. It only requires adding rules to the predetermined grammar, and re-exporting the parser. If the application were extended to handle different type of emails or to take in more details, changes in the backend might necessitate a major update. To understand what grammar and details the user would want to be included I would try a forum or questionnaire to those who use and want to help develop the tool, allowing them to submit any changes they think would be useful to have, each one will be evaluated for both usefulness and the potential impact on the existing system.

Despite this, I still think it is important to keep a fairly limited grammar otherwise it will end up being too confusing for a novice user to follow the rules, along with becoming overly complicated in the codebase. For a much wider grammar it is better to use a tool like chatGPT, where instead of using templates it uses a generative language model to create the email Section 2.3.1

6.3 User interface

When conducting the user testing one participant made a comment that it was slightly off-putting that the theme of Thunderbird did not match that of the Write Right tool (see Section 5.1.3). This is something that I think would improve the user interface and also make the tool more accessible. As if a user has visual impairments like colour blindness a specific theme might make the text more discernible. I have done some research into how to achieve this, Thunderbird developers are currently developing an experimental API that allows themes information to be obtained and updated.

It would also be beneficial in the future to conduct AlB testing on a larger sample size (n=50+). Where different user interfaces could be tested to see which is the easiest to use but also the fastest for the user to go through. I was unable to conduct any significant AlB testing during this project as I was more focused on the functionality of the tool.

From reading more user design articles I discovered that the user warning system currently implemented could be improved. Nomensa discusses the do's and don'ts for user warning messages. I would change the underlying user warning system so that it warns the user at the point where they made the error, as opposed to having error messages for the form at the bottom of the page.

6.4 Improving user experience

Some steps that could be taken to improve the user experience when using the extension would be to allow for more shortcut tools when using the extension, for example, the suggestion box items could have hotkeys associated with them so that a user doesn't have to go between the keyboard and the mouse. I would also like to extend the context menu user experience. Right now it only has the ability to convert the text from the compose window into a default template for either a question or a meeting. Since the default template object Section 4.5.1 is designed to be modular and be able to be built upon or miss details out and still make sense, I think it could benefit the user to have access to these without having to write a full email. For example, if a user wanted to write a different kind of email but still wasn't sure how to start they could use an item on the context menu called "Insert Introduction", and from Section 4.5.1 the intro field would be added to the compose message but would not remove or change any of the text already added in. Shortcuts could also be used here to implement the same functionality, this could be set in the options page.

6.5 Locales

The template module (Listing 2) is designed to be able to extend the functionality for different locales working in the future. To do this when implementing the templates module I set it up so that in the future it would be able to include other languages without needing to change the actual code. This will work okay for adapting new templates in other languages, however, this won't allow different text to be entered into the parsing section, nor will it update any of the labels for the user interface popup.

The extension is set up to take the user's locale in, it requests this permission in the manifest, yet it is not implemented to work with any aspects of the system yet, so some refactoring will be required to get this to work. To show the labels on the elements on the popup, the page will need to be refactored to use key stores as opposed to hard-coded strings. This will both allow users with different locales to be able to understand the user interface but will also help with the upkeep of the code and the extension as it will make it easier for future developers to change a value. This helps also reduce code duplication and helps follow the don't repeat yourself coding principle.

Chapter 7

Conclusions

7.1 Project summary

The Write Right tool is an innovative Thunderbird extension that enables users to quickly and easily compose detailed and well-formatted emails for organizing meetings or communicating with teachers. By extracting details from short text inputs and converting them into customizable templates, the tool streamlines the email-writing process and ensures that users include all necessary information without being overwhelmed. Through a rigorous development process that involved creating detailed extraction algorithms using plain Javascript, the tool has been refined to provide high levels of accuracy and customization. User testing has shown that the tool is both effective and user-friendly, with all participants who took part in the user testing being able to successfully create emails using the tool's default or custom templates. These results demonstrate the potential of the Write Right tool to aid students and allow them to write emails both better and faster.

7.2 Skills and knowledge developed through the project

During the early stages of the project, I had only basic knowledge of natural language processing (NLP) from an introductory course I had taken. However, as the project progressed, I developed more advanced skills, such as creating a parser using parsing expression grammar and using different technologies. This was a significant growth area for me, as I had never worked with NLP in JavaScript before.

Throughout the project, I learned the importance of planning and organizing code, which helped me improve my skills in writing clean and maintainable code. Before this project, I had a tendency to use a code-first approach, which often resulted in writing unnecessary code and creating technical debt. However, by planning out the project's structure and organizing the code into modules, I was able to write more efficient and organized code. This was a valuable lesson that I plan to apply in future projects.

Finally, the project provided me with valuable experience in web development, particularly in creating an extension or add-on. While I had been interested in web development since a young age, this was my first time creating an extension, which taught me about working with pre-defined systems and the limitations of web development. Additionally, I gained an understanding of the differences between extensions and typical web pages, such as limited user and machine access. This experience broadened my knowledge of web development and will be useful in future web development projects.

During the project, I also learned about the importance of design and its impact on the user experience. I discovered that simple designs are often more intuitive and userfriendly than complex designs, which can be overwhelming for users. By adopting a simple design approach, I was able to create a more user-friendly interface for the Write Right tool. This experience taught me that design is an essential aspect of any project and that investing time in designing user-friendly interfaces can greatly improve the overall user experience.

7.3 Reflections

Overall, I believe I was successful in creating a tool to help students write well-formed emails to their teachers. All the responses from the user testing survey said that they thought the tool both improved the formality and the speed they write emails, which were the major goals of this project.

If I were to do this project again I would try to understand the Thunderbird documentation earlier on. This would allow me to be able to put more realistic estimations on how long certain functionality would take to implement. I would also conduct user testing much earlier on with a less complete version to try and understand more about what the user wants from the tool, instead of adding and removing grammar and other functionality throughout the process.

References

Ford, Bryan, 2004. Parsing expression grammars: a recognition-based syntactic foundation. *Sigplan not*. [Online], 39(1), pp.111–122. Available from: https://doi.org/10.1145/ 982962.964011.

Bird, Steven, Klein, Ewan, and Loper, Edward, 2009. *Natural language processing with python: analyzing text with the natural language toolkit.* O'Reilly.

Stephens, Keri K., Houser, Marian L., and Cowan, Renee L., 2009. R u able to meat me: the impact of students' overly casual email messages to instructors. *Communication education* [Online], 58(3), pp.303–326. Available from: https://doi.org/10.1080/03634520802582598.

Jarrett, Caroline and Gaffney, Gerry, 2010. *Forms that work: designing web forms for usability*. Elsevier/Morgan Kaufmann.

Majda, David and Ryuu, Futago-za, 2010. *Pegjs* [Online]. Available from: https://pegjs. org/.

Manning, Christopher, Surdeanu, Mihai, Bauer, John, Finkel, Jenny, Bethard, Steven, and McClosky, David, 2014. The stanford corenlp natural language processing toolkit. *Proceedings of 52nd annual meeting of the association for computational linguistics: system demonstrations* [Online]. Available from: https://doi.org/10.3115/v1/p14-5010.

Mascarenhas, Fabio, Medeiros, Sérgio, and Ierusalimschy, Roberto, 2014. On the relation between context-free grammars and parsing expression grammars. *Science of computer programming* [Online], 89, pp.235–250. Available from: https://doi.org/10.1016/j. scico.2014.01.012.

Economidou-Kogetsidis, Maria, 2015. Teaching email politeness in the efl/esl classroom. *Elt journal* [Online], 69(4), pp.415–424. Available from: https://doi.org/10.1093/elt/ccv031.

Tirumala, Sreenivas Sremath, Ali, Shahid, and Babu, Anjan, 2016. A hybrid agile model using scrum and feature driven development. *International journal of computer applications* [Online], 156(5), pp.1–5. Available from: https://doi.org/10.5120/ijca2016912443.

Education Scotland, 2017. *Benchmarks first level all curriculum areas*. Available from: https://www.education.gov.scot/nih/Documents/FirstLevelBenchmarksAllAreas_.pdf.

REFERENCES

Thomas, David and Hunt, Andrew, 2019. *The pragmatic programmer: your journey to mastery, 20th anniversary edition* [Online]. 2nd. Addison-Wesley Professional. Available from: https://dl.acm.org/doi/book/10.555/320326.

Xu, Tammy, 2020. *What are tracer bullets in software development?* Available from: https://builtin.com/software-engineering-perspectives/what-are-tracer-bullets.

Thesing, Theo, Feldmann, Carsten, and Burchardt, Martin, 2021. Agile versus waterfall project management: decision model for selecting the appropriate approach to a project. *Procedia computer science* [Online], 181, pp.746–756. Available from: https://doi.org/10. 1016/j.procs.2021.01.227.

Popup forms: bad practice or best practice?, 2022. Available from: https://www.formasse mbly.com/blog/popup-forms/.

Richman, Danny, 2022. *Tweet from @DannyRichman*. https://twitter.com/DannyRichman/ status/1598254671591723008. [Accessed March 11, 2023].

Salazar, Kim, 2022a. *Evaluate interface learnability with cognitive walkthroughs* [On-line]. Available from: https://www.nngroup.com/articles/cognitive-walkthroughs/.

Salazar, Kim, 2022b. *How to conduct a cognitive walkthrough workshop* [Online]. Available from: https://www.nngroup.com/articles/cognitive-walkthrough-workshop/.

Sheng, Alice, 2022. *The difference between chinese and british business practice: digital marketing agency: easy marketing*. Available from: https://easymarketing.uk/the-difference-between-chinese-and-british-business-practice/.

Jonathan Perry-Houts, Jonathan Kamens, 2023. *Sendlater extension for thunderbird* [Online]. Available from: https://addons.thunderbird.net/en-us/thunderbird/addon/send-later-3/.

Milmo, Dan, 2023. Chatgpt reaches 100 million users two months after launch. *The guardian* [Online]. Available from: https://www.theguardian.com/technology/2023/feb/02/chatgpt-100-million-users-open-ai-fastest-growing-app.

1292701 - autoclose popups shouldn't close when they open a modal dialog (e.g., file picker), n.d. Available from: https://bugzilla.mozilla.org/show_bug.cgi?id=1292701.

Nomensa, n.d. *Ux design: four ways to display error messages* [Online]. Available from: https://www.nomensa.com/blog/ux-design-four-ways-display-error-messages.

The collaborative interface design tool. N.d. Available from: https://www.figma.com/.

Thunderbird, n.d. Available from: https://developer.thunderbird.net/add-ons/about-add-ons.

Thunderbird addon code knowledge base, n.d. Available from: https://cleidigh.github.io/ ThunderKdB/xall/extension-list-all.html.

Participant Information Sheet

Project title:	Tool to help students write "proper" emails
Principal investigator:	Brian Mitchell
Researcher collecting data:	Angus Paroles ICIX A
Funder (if applicable):	

Participants' information sheet

This study was certified according to the Informatics Research Ethics Process, RT number 422851. Please take time to read the following information carefully. You should keep this page for your records.

Who are the researchers?

Angus Parsons, s1942287@ed.ac.uk

Brian Mitchell, brian.x.mitchell@ed.ac.uk

What is the purpose of the study?

To see how accessibly and intuitive the tool is, whether users who have never seen the tool before are able to use it and complete tasks with the tool.

Why have I been asked to take part?

The research target group is students, primarily those who are not from United Kingdom or who's first language isn't English.

Do I have to take part?

No – participation in this study is entirely up to you. You can withdraw from the study at any time, up until completing and submitting the survey without giving a reason. After this point, personal data will be deleted and anonymised data will be combined such that it is impossible to remove individual information from the analysis. Your rights will not be affected. If you wish to withdraw, contact the PI. We will keep copies of your original consent, and of your withdrawal request.

What will happen if I decide to take part?

39

Appendix B

Participant Consent Form

Project title:	Tool to help students write "proper" emails	
Principal investiga	cipants' consent form	
Researcher:	Angus Parsons	
PI contact details:	brian.x.mitchell@ed.ac.uk	

By participating in the study you agree that:

- I have read and understood the Participant Information Sheet for the above study, that I have had the opportunity to ask questions, and that any questions I had were answered to my satisfaction.
- My participation is voluntary, and that I can withdraw at any time without giving a reason. Withdrawing will not affect any of my rights.
- I consent to my anonymised data being used in academic publications and presentations.
- I agree to completing an anonymised survey about the project.
- I understand that my anonymised data will be stored for the duration outlined in the Participant Information Sheet.

Please tick yes or no for each of these statements.

- 1. I allow my data to be used in future ethically approved research.
- **2.** I agree to take part in this study.





Name of person giving consent	Date dd/mm/yy	Signature	
Name of person taking consent	Date dd/mm/yy	Signature	



40

Appendix C

Sign-Up Sheet



ኤ

Sign-op Sheet

User Study Sign-up Sheet

This form is to say you might want to take part in one (or more) user studies to help final year students develop tools to help students. These tools might be useful to you in your future studies.

All UoE students are welcome to participate in the user studies. You are equally valuable whether you are new to Computer Science or experienced. Indeed the greater the diversity, the better. This includes native and non-native speakers of English, and people with accessibility requirements or impairments, especially any affecting the way you use computers.

User studies might include being asked to test some software, to say what you'd like the software to do or how it should do it, or to tell your experience in the topic, if any. No specific experience is necessary to participate in the studies: everyone from "beginner" to "experienced" is equally welcome.

The format of user studies may differ per project. You are likely to be asked to participate in 2 or 3 sessions per project you express an interest in.

Submitting this form does **not commit you** to definitely participating and you may withdraw later.

Any information you do submit is confidential and will be deleted once sign-ups have been processed.

Disclaimer: the form's signature image may or may not be indicative of what happens during any user study. Any similarity between the events depicted in the signature image and he user study may or may not be entirely coincidental.

* This form will record your name, please fill your pame.

Eligibility check

1. I am a current University of Edinburgh student *



🔵 No

Sorry: these user studies are only for current University of Edinburgh students

Details

- 2. The following projects are available for you to participate in. The students doing the projects want to hear from beginners and novices as well as people with relevant experience.
 - Tool to help manage timetables and workloads this tool is for everyone but anyone with accessibility requirements is especially encouraged to participate
 - 2. Tool to help you **check your assignment complies** with submission guidelines this might save you some stress and maybe a few marks
 - Tool to help you write "proper" emails to university staff
 this is as much for native English speakers as it is for nonnative English speakers
 - 4. Tool to help you **learn command-line programming** this type of programming helps with task automation and processing lots of files
 - Tool to help turn program code into natural language (for example English, Spanish) — this is as much for native English speakers as it is for non-native English speakers, and for experienced programmers as it is for beginners
 - Tool to help managing data and experiments in Machine Learning — you don't have to any experience with Machine Learning

Are you interested in each project? *

	l don't know	Not interested	Interested
Managing timetables and workloads	\bigcirc	\bigcirc	\bigcirc
Checking assignment compliance	\bigcirc	\bigcirc	\bigcirc
Managing Machine Learning data and experiments	\bigcirc	\bigcirc	\bigcirc
Learning command- line programming	\bigcirc	\bigcirc	\bigcirc
Turning program code into	\bigcirc	\bigcirc	\bigcirc

3. Please specify the **one or two** projects you are **most** interested in testing. *

Please select at most 2 options.

Managing timetables and workloads
Checking assignment compliance
Writing emails to university staff
Learning command-line programming
Turning program code into natural language
Managing Machine Learning data and experiments

About you

These questions are here so we can see the range of diversity.

We respect your answers and we respect your right not to answer.

4. Your Student ID (without the S) — also known as your UUN *

- 5. Are you a native speaker of English or has English been the main language you have used in studying? *
 - Do not wish to say
 No
 Yes
- 6. Do you have any kind of accessibility or educational requirements that affect the way you use a computer?

We will not ask you for any details. You may volunteer this information to the project student(s) at a later date if and only if you want to. *

Do not wish to say

- Yes but I do not like to admit I do
- Yes and I openly admit I do, or it is impossible to disguise my situation
- 🔵 No
- 7. If you have any questions or comments please put them here.

Please make it clear which project each question or comment is intended for

intenueu ior.

Confirmation

- 8. Expressions of interest will be processed in batches so there might easily be a delay of several weeks before you receive a response *
 - I understand there might be several weeks before there is a response. By submitting I agree to be contacted via my University email by the student(s) whose project(s) I have expressed an interest in.

This content is neither created nor endorsed by Microsoft. The data you submit will be sent to the form owner.



Pre User Testing

Appendix D Pre User Testing

Pre-test Questionnaire

1. What year of university are you in?

Mark only one oval.

1

____4

- 5
- 2. What do you study?

3. Are you an international student?

Check all that apply.

__ Yes No

4. Is English your first language?

Check all that apply.

Yes

5. What is the main reason(s) you email a teacher/lecturer?

12/04/2023.	13:43
12/01/2023,	10.10

Pre User Testing

6. Were you ever nervous about sending emails to teacher/lecturers?

Mark only one oval.

O Yes

🔵 No

- 7. If the answer was yes to the previous why?
- 8. How frequently do you send emails to teachers?

Mark only one oval.

🔵 Daily

Weekly

Monthly

Very rarely

9. What would you want from a tool that helps student write emails to teachers/lecturers?

This content is neither created nor endorsed by Google.

Google Forms

After Test

Appendix E

After Test

Post-test Questionnaire

1. Did you find the tool useful

Mark only one oval.

Yes

2. Did you find the grammar rules easy to understand

Mark only one oval.

____ Yes ____ No

3. How often would you use this tool?

Mark only one oval.

	Never again
1	
2	\bigcirc
3	
4	
5	
	Every email

12/04/2023, 13	:48	After Test
4.	Do you think it is obvious what inputs the user must fill out?	
	Mark only one oval.	
Yes		
	No	
	Other:	

5. Do you think the email types are suitable?

Mark only one oval.

C	\supset	Yes
\subset	\supset	No

Maybe

6. Were you able to complete all tasks if not which ones could you not?

Mark only one oval.

Yes		
No		
Other:		

7. Do you think this improved the etiquette in the emails you write to lecturers?

Mark only one oval.

── Yes

____ No

____ Maybe