Project Proposal: Toolkit for an Abstract Wikipedia

Matthew Craig
University of Cape Town
Cape Town, South Africa
CRGMAT002@myuct.ac.za

Jordy Kafwe
University of Cape Town
Cape Town, South Africa
KFWJOR001@myuct.ac.za

Toolkit for an Abstract Wikipedia
30 April 2024

1 Abstract

A vision for a truly multilingual Wikipedia has been proposed [18], architected [20], and specified [2, 5]. This vision, titled Abstract Wikipedia, aims to generate Wikipedia articles from base, abstract representations of content. Past research [2, 16, 20] has shown the endeavour's viability, however, work is still needed if a complete system is to be actualised. This proposal details a set of software tools that broaden the accessibility and functionality of Abstract Wikipedia. This includes a tool for managing templates and a tool for extracting content for use in constructors.

2 Introduction

Abstract Wikipedia is a much broader project, of which this project forms only a small part. Abstract Wikipedia envisions a system of open collaboration between people of diverse, multilingual backgrounds [20]. Its goal is to leverage *Wikidata* [19] and *Wikifunctions* [20] to facilitate the creation of language-agnostic representations of content. Natural Language Generation (NLG) techniques will be utilised to produce articles from these abstract representations in a vast array of languages.

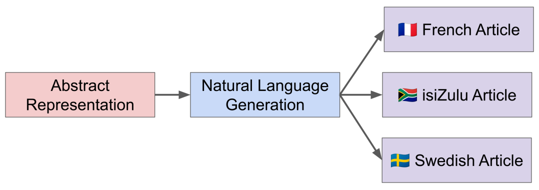


Figure 1: Abstract Wikipedia Concept Diagram

Wikidata is an open, collaborative knowledge base [19]. Wikidata houses a large collection of labelled entities and the relationships they have with other entities (figure 4). Its goal is to provide a diverse collection of machine-readable knowledge that anyone may contribute to and benefit from [19]. The content is, however, largely inaccessible to the broader public. This is due to the technical barrier prohibiting interaction with the content. Abstract Wikipedia intends to solve this problem by producing human-readable natural language from the knowledge housed in Wikidata.

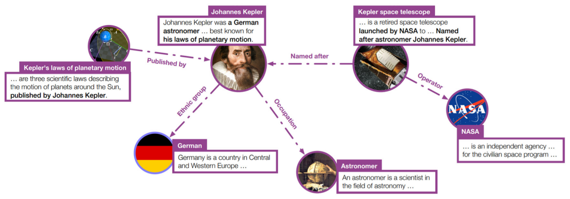
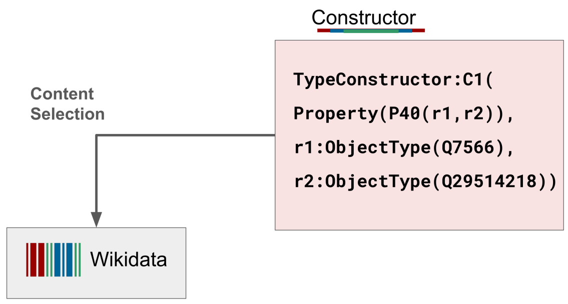

Johannes Kepler
Johannes Kepler was a German astronomer … best known for his laws of planetary motion.
Kepler space telescope
… is a retired space telescope launched by NASA to … Named after astronomer Johannes Kepler.
Kepler's laws of planetary motion
… are three scientific laws describing the motion of planets around the Sun, published by Johannes Kepler.
Published by
Named after
Operator
Ethnic group
Occupation
NASA
… is an independent agency … for the civilian space program …
German
Germany is a country in Central and Western Europe …
Astronomer
An astronomer is a scientist in the field of astronomy …

Figure 2: Wikidata Knowledge Graph [21]

The abstracted representations of content are given the term *constructors* [20]. Constructors are declarative statements of content to be selected from Wikidata. The declarations can be conceptualised as expressive arrangements of language-independent Wikidata identifiers. They are modular representations of content that can be composed to form an article. A modelling language for constructors, *CoSMo*, has recently been specified [2].



```
TypeConstructor:C1(
Property(P40(r1,r2)),
r1:ObjectType(Q7566),
r2:ObjectType(Q29514218))
```

Figure 3: Example CoSMo [2] Constructor Declaration

Each constructor is to have, associated with it, a set of language-specific templates for generating natural language. Each template describes how constructor-specified content is to be arranged in a particular language [5]. Templates are composed of slots interspersed with free-text [5]. Slots can represent interpolations of arguments, lexical function calls, or sub-templates [5]. Slots are to be given dependency labels which identify their grammatical role [5]. Free-text is simply a string of text in the chosen language. Natural language is to be generated from the realisation of these templates. In figure 4, slots are denoted by braces and free text is rendered in green.

```
Age_renderer_sv(Entity, Age_in_years):
  Person(Entity)} är {Age_in_years} år gammal ."
```

Figure 4: Example Template [5]

Templates act as an intermediary representation between constructors and natural language. The templates effectively act as functions that take in content from a constructor and return natural language. The existence of templates will, ideally, facilitate the rapid production of informationally-consistent content across many languages. The re-usability and composability of templates are key to this prospect.

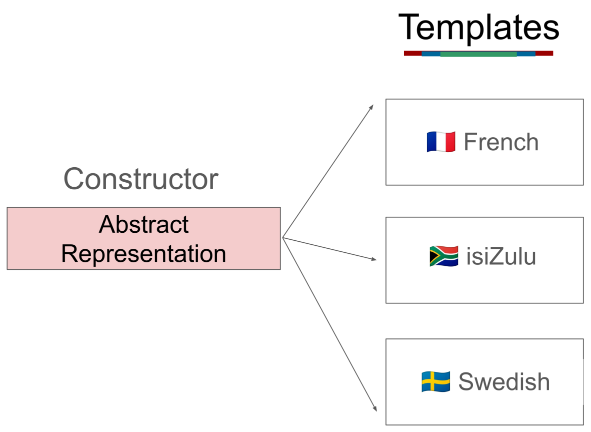


Figure 5: Relationship between Constructor and Templates

3 Problem Statement

The scope of this proposal's project does not seek to address all the problems that Abstract Wikipedia aims to. Rather, the components intend to alleviate Abstract Wikipedia's current shortcomings. These can be loosely categorised into two categories: Functionality and Accessibility.

3.1 Problem: Functionality

The functionality required by Abstract Wikipedia is largely absent. Key components such as the constructors and templates have not been developed. While partial implementations - notably Ninai/Udiron [16] - have been attempted, progress towards a production-ready system has not yet begun. There are no existing tools capable of providing the functionality necessary to achieve Abstract Wikipedia's goals.

3.1.1 Constructors

There is, presently, no software implementation of constructors. This proposal gives particular concern to the current absence of content selection and Wikidata integration [20]. Without the functionality of content extraction from Wikidata, the constructors are unable to fulfill their intended purpose within the system.

3.1.2 Templates

Similarly to constructors, an implementation of templates does not yet exist. This proposal focuses on the issue of absent template validation and parsing. [5]. Templates are unable to be realised as natural language without a system that can handle these tasks.

3.2 Problem: Accessibility

Significant technical barriers are preventing the general public from contributing to and benefiting from Abstract Wikipedia. Even if constructors and templates were to be fully functional in selecting and realising content, there persist significant hurdles in using them. There are no existing tools that improve the ease-of-use, accessibility, or comprehensibility of the project.

Two aspects of accessibility, *content awareness* and *template creation*, are specifically relevant to this proposition.

3.2.1 Content Awareness

Creating a constructor requires awareness of the relevant Wikidata entries that are to be used. Currently, such discoverability is relegated to manual interaction with Wikidata's SPARQL endpoint [19]. Ideally, a system would exist that suggests relevant content to a user during the constructor-creation process.

3.2.2 Template Creation

The template-creation process is complex and unintuitive without guidance. This deters a majority of potential users. It is infeasible to expect users to devote time to understanding the syntax and functionality of templates.

4 Project Aims

The project proposed here aims to solve many of the problems that currently affect Abstract Wikipedia. To solve these problems, this project includes the development of:

a) *CRAFT*: A system for optimally extracting relevant Wikidata content.

b) *TempTing*: A web tool that streamlines the management and creation of templates.

Both components are to be developed in parallel with another project which aims to develop a tool for managing the creation of constructors. Ideally, in future, all 3 components will be integrated. At present, however, this falls outside of the proposal's scope.

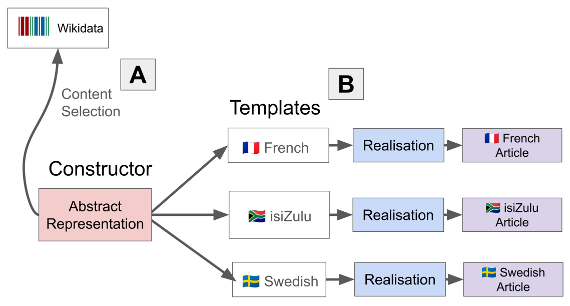


Figure 6: Project Components
A: *CRAFT*
B: *TempTing*

4.1 CRAFT

CRAFT (Content Retrieval and Finder Tool) will interface directly with Wikidata to ensure that the desired content is efficiently retrieved. Wikidata exposes a SPARQL endpoint for interfacing with its knowledge graph [19]. The tool will take a constructor statement as input, parse the statement and fetch the specificied content. The tool will also take a Wikidata identifier or a natural language text as input. From either input, the tool must determine the necessary SPARQL or SQL query for retrieving the entity and related data. The system will provide an API through which a hypothetical constructor-management tool may interface. This component aims to address the following problems:

(1) *Functionality*: Currently, constructors are not actualised; their intended purpose of content selection is unfulfilled.

(2) *Accessibility*: A user wishing to create a constructor has no way of knowing what data is available without manual exploration of Wikidata.

4.2 TempTing

The tool for managing templates is intended to be user-facing. The tool will allow users of various languages to create templates necessary for the realisation of constructors in natural language. The tool intends to assist in the production of these templates through a web interface. The primary functionality of the web app will include both a textual and a visual diagramming tool for constructing templates. The templates should be parsed such that they can be converted between either representation. It has been concluded that realisation (NLG) falls outside of the scope of this project. Instead, TempTing will prioritise the robustness and accessibility of the template creation process. The TempTing tool should ideally achieve:

(1) *Functionality*: A first step towards the successful implementation of a template-realisation system.

(2) *Accessibility*: The template creation process is sufficiently abstracted and simplified.

5 Procedures and Methods

5.1 Development Platform and Architecture

5.1.1 CRAFT

The content retrieval tool will be implemented as a RESTful API. The RESTful API backend will be written in FastAPI, a Python framework for building web APIs.[17] Python was chosen due to it being one of the supported languages in Wikifunctions.[15]

This API backend will interface with the SPARQL endpoint provided by the local Wikibase, running in Docker, as well as query the MariaDB relational database used by this Wikibase [19]. Wikibase is the software platform that powers Wikidata, enabling the management of its large structured knowledge base. It allows one to run a local or self-hosted Wikidata[19]. The query expansion logic will be implemented as part of this backend. That is, after receiving the user input, the tool will convert it to the appropriate SPARQL or SQL query and then return the data as JSON in an HTTP response. To overcome the 60-second timeout on the public Wikidata SPARQL endpoint, a subset of Wikidata will be created from the publicly available data dumps [14]. WDumper, a no-code tool designed to extract specific subsets of data from Wikidata dumps will be used to do this [1]. The resulting Wikidata subset will then be used to populate the local Wikibase copy.

5.1.2 TempTing

The tool's architecture will be split between a backend and a frontend. The backend will handle template parsing and storage. The frontend will include the web app for interfacing with the system. The web app will contain both a textual and visual interface for template creation.

Frontend
The frontend will be developed in Svelte, a Javascript framework [6]. The visual diagramming tool will be developed with SvelteFlow, a Svelte-compatible diagramming library [11]. The library enables the creation of dynamic, interactive diagrams. These diagrams produce JSON output [11] that can be feasibly parsed into an abstract syntax tree.

The text-editor interface will be developed using the CodeMirror [8]. This Javascript library will simplify and accelerate the development of text-editing features. It has support for syntax highlighting, code-completion, and linting [8]. These features can be achieved by connecting the library to the abstract syntax tree produced by the parser.

Backend
The backend, including the parser, will be developed in Javascript via the full-stack SvelteKit framework [7]. Javascript was chosen as it is a supported language on Wikifunctions [15]. This facilitates future parser interoperability with Wikfunctions.

5.2 Implementation Strategy

The project will adopt an iterative waterfall method implementation strategy. This approach combines the structured phases of the waterfall model with the flexibility to iterate within each phase. The specific tasks found in each phase can be found in the Gantt chart in section 8. This will offer the clear direction of Waterfall, while still allowing leeway for feedback accommodation. A full implementation of agile has, however, been avoided. It was deduced that there is not enough time or opportunity to continuously evaluate with stakeholders.

5.3 Expected Challenges

5.3.1 CRAFT

This project faces multiple challenges. First, accurately determining the "relatedness" of Wikidata content can be difficult, as there might not be a single perfect metric. Implementing query expansion for the relational MariaDB database can be complex. Unlike SPARQL, which is specifically designed for navigating relationships within RDF data like Wikidata, SQL's capabilities for expanding queries to explore connected information are more limited. [19] This could restrict the comprehensiveness of the extracted content.

5.3.2 TempTing

Understanding and defining the template syntax is likely to be challenging. There currently are very few examples [5] and edge case handling may likely be unclear. Developing a concrete grammar for the parser will require an in-depth understanding of the intentions and features of the syntax. Obtaining knowledge about an ideal parsing approach will be difficult, as few resources are geared towards natural language generation. Integrating the parsing with CodeMirror (the text editor library) will be challenging, as CodeMirror recommends a bespoke format for grammar specification [5].

A prominent challenge will be to define a compatible visual language for templates. This visual language will need to balance expressiveness with ease-of-use. If this is later deemed infeasible, a contingency plan has been made to exclude the visual representation of templates.

5.4 Evaluation

5.4.1 CRAFT

Unit testing will be used to ensure the correctness of individual code modules. That is, the individual backend components like functions and modules will be isolated for focused testing. This includes functionalities for interacting with the local Wikibase instance. Mocks or stubs can be used to simulate the Wikibase environment for controlled testing. Unit tests will also target the core logic of converting user queries into SPARQL or SQL statements. Specific user's inputs will be provided and then it will be checked that the generated queries accurately reflect the user's intent. This comprehensive unit testing strategy targeting critical backend components will, ideally, identify any errors in the individual modules.

5.4.2 TempTing

The tool is focused on accessibility and ease-of-use, thus, user testing is essential for determining that goals have been met. Participants will be asked to navigate the interface and achieve tasks without prompting. This will be executed as time-limited screen recordings. Afterwards, opinions will be given in an anonymous questionnaire/survey format. Questions will include "How easily could you achieve x task?" and "How frustrating was feature y?". The targeted user group will be existing Abstract Wikipedia, Wikidata, and Wikifunctions contributors. The number of participants surveyed will be kept small (2 to 5 users). A significant window will be dedicated to this process. This is to avoid any unforeseen issues that arise from a dependency on the small pool of potential participants. Additionally, heuristics evaluation will be performed to ensure a more objective measurement is included.

During the planning phase, a series of test templates and their expected parsed structures will be defined. These can later be used to validate that the parser is behaving as expected. Key parts of the code will also be tested with unit testing. To facilitate this, the code will be structured in a modular manner, primarily as pure functions. This has the added benefit of streamlining future Wikifunctions transposition.

Bidirectional testing between textual and visual templates will be carried out. This will ensure that the visually constructed templates are analogous to their textual counterparts.

6 Anticipated Outcomes

6.1 Key Features

6.1.1 CRAFT

CRAFT will exist as a RESTful API, accessible through an HTTP endpoint. Firstly, it will process constructors. This involves accurately extracting Wikidata identifiers from constructors. Secondly, it will implement a query generation module that converts Wikidata identifiers into the appropriate SPARQL or SQL query. This query generation module will also implement query expansion to suggest items related to the target. Therefore the key features are:

(1) Processing Constructors.

(2) Production of Queries.

(3) Interfacing with Wikidata.

(4) Providing access though an API Endpoint.

6.1.2 TempTing

TempTing will exist as a web app with support for template selection and management. This must include tools facilitating both visual and textual template creation. Visual creation will be done through a diagrammatic, drag-and-drop, visual interface. This should abstract the process of creating templates. A hypothetical example of a visually produced template can be seen in figure 7. Textually, the tool should include features such as syntax-highlighting, linting, pretty printing, and auto-completion. A hypothetical example of template syntax highlighting can be seen in figure 4

The tool should have parsing functionality, including bi-directional conversion between visually-created templates to their textual equivalents. Additionally, this should allow users to verify a template's adherence to the syntax requirements.

Thus, TempTing's three key features are:

(1) A visual template diagramming tool.

(2) A template text editor.

(3) Parsing for bi-directional conversion.

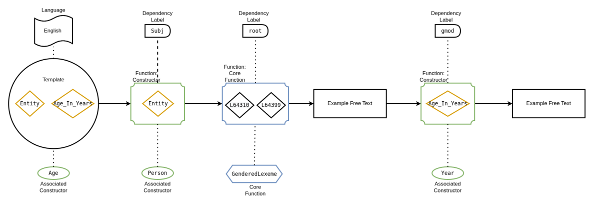


Figure 7: Hypothetical Visually Created Template

6.2 Impact

The core problems of functionality and accessibility will be addressed. This will push Abstract Wikipedia forward and spark new interest in its progression.

6.2.1 Functionality

Providing a base of functionality will allow future development to expand upon the work done here. This will spur research and engagement with Abstract Wikipedia. Solving the content selection problem will mean that constructors may viably be integrated into a complete system. Successful template validation and parsing will ideally be a precursor to the realisation process.

6.2.2 Accessibility

Improved accessibility will enable a far greater pool of contributors to engage with Abstract Wikipedia. This will significantly impact the public exposure that the project receives. CRAFT will improve the discoverability of Wikidata content, assisting in the creation of constructors. TempTing's implementation will reduce the complexity of the template creation process.

6.3 Key Success Factors

Each of the introduced components aims to address a set of core requirements. The key factor determining the project's success, thus, will be the degree to which these requirements are met.

6.3.1 CRAFT

CRAFT's success is dependent on three main factors. Firstly, accurately parsing constructors to determine what data to fetch. This influences all subsequent features after processing the received constructor. Achieving this will thus be determined by whether an algorithm to process a constructor of the Wikidata identifiers it specifies can be developed.

Secondly, the generation of queries must adequately and correctly fetch desired data. This is important as correctness is imperative in an encyclopedia, like Wikipedia [2].

Lastly, it must be ensured that query expansion, for both SQL and SPARQL queries, is accurate. This enriches the retrieved data by providing related entities and properties. To broaden the scope of information, the expanded data must be relevant.

Therefore, the produced tool must meet these core requirements to be considered successful. However, the inclusion of niceties such as caching and further optimisations would be preferable.

6.3.2 TempTing

The principal requirements affecting TempTing relate to the parsing of templates. Both visually and textually created templates need to be parsable to a base abstract syntax tree (AST) [3]. This will facilitate validation and bi-directional conversion between either form.

The issue of parsing can be more easily navigated by reframing it as a well-established problem. Template parsing is highly comparable to the development of a domain-specific language (DSL) [3]. Ergo, achieving this requires the development of a lexer and parser [3] for the template syntax. Following this conventional approach will yield additional benefits. The path to achieving the desired linting, syntax highlighting, and auto-complete features is demystified in this framing. Parsing can be performed efficiently and concisely; the template language is governed by a context-free-grammar [5].

Parser implementation will require formalisation of the grammar such that parsing rules are understood. The parser will either be implemented via recursive descent parsing [3] or combinatoric parsing [9]. The decision will depend on the most natural expression of the grammar.

TempTing will be considered successful if it can meet these parsing requirements. Ideally, however, all features listed in section **??** will be achieved.

7 Ethical and Legal Issues

7.1 Licensing and Credit

All libraries and frameworks planned for use in this project are open source and use permissive licences (either MIT or BSD). This means that they may be used freely in our project without legal requirement for credit. However, all usages of external code and libraries will be made explicit to distinguish which aspects of the work are our own. Each library planned for use, its licence, and its source code can be found in table 1.

Library	Licence	Source Code
Svelte	MIT	[6]
SvelteKit	MIT	[7]
SvelteFlow	MIT	[11]
shadcn-Svelte	MIT	[10]
Tailwind	MIT	[22]
CodeMirror	MIT	[8]
FastAPI	MIT	[17]
Universal Dependencies	Apache	[23]

Table 1: Software Licences

It is essential that any work benefited from or utilised in the project be credited. The template tool makes strong use of the template syntax defined on the MediaWiki Metawiki [5]. The template syntax is also defined in terms of dependency labels sourced from Universal Dependencies [23]. This project is obliged to highlight all instances in which the work of others is used.

Abstract Wikipedia, and more broadly Wikipedia, aims to allow people of diverse languages to "collaborate in the sum of all Knowledge" [18]. Keeping with this spirit, this project has decided to open source all of its code. Open-sourcing the code will allow future work to expand upon any achievements made here. The code will be hosted on GitHub and licensed under the permissive MIT licence.

7.2 Participants

As discussed previously, the template tool's development will involve the surveying of participant questions. Care must be taken to ensure all data is anonymised, participants give informed consent, and participants do not feel forced to take part. Efforts should be made to clearly explain the rights participants have when engaging in the study. The participants should understand what data is being captured and how it is being used. In alignment with Abstract Wikipedia's vision for a multilingual Wikipedia, it will ensure that a linguistically diverse group is selected and fairly represented.

7.3 Content Selection Accuracy

It is essential that content is accurately selected from Wikidata. Inaccurate selection may result in content that includes misleading falsehoods. Presenting false content as factual could result in harmful misunderstandings among users. For example, an article incorrectly portraying someone as having committed violent acts would be highly libellous. The accuracy of CRAFT's selection is, thus, of ethical concern.

7.4 Related Work

7.4.1 Architecture and Design

The project will be developed with heavy influence from the work of Denny Vrandečić. Vrandečić justified [18] and proposed [20] the Abstract Wikipedia project.

The constructors were given a far stronger, formal specification in CoSMo [2]. CoSMo help guide the requirements of CRAFT. The template syntax, used by TempTing, has also been specified [5]. TempTing's parser will be designed such that it strictly follows this guideline.

7.4.2 Related Implementations

Several past implementations are similar to the project proposed here. Nina/Udiron [16], in particular, is a promising attempt at implementing an NLG system for Abstract Wikipedia. Ninai/Udiron does not, however, make use of templates in the same way as this project.

An implementation of the templates has been conducted [4]. This implementation is not fully featured and lacks the accessibility benefits this project proposes. The proposed visual diagramming tool will help solve this problem and will take heavy influence from the work done for the ToCTEditor [13].

The participant evaluation procedure will be influenced by the work conducted in ArticlePlaceholder [12].

The CoSMo specification paper established that current content selection tools often require considerable technical expertise, hindering accessibility [2]. Existing tools, such as TexToData, offer some multilingual capabilities, but they rely on translation methods that are not inherently multilingual [2]. In addition, while graphical query builders like VSB and RDF Explorer facilitate query construction, they lack modularity and multilingual support [2]. Furthermore, they do not accept conceptual data models, which are necessary for a constructor implementation. [2]. These limitations, therefore, make it necessary to develop a novel content selection tool for use in Abstract Wikipedia.

8 Project Plan

See appendix for Risks, Timeline, Deliverables, and Milestones.

8.1 Work Allocation

The stated software components are kept largely independent in their development to maximise parallelisation in the development process.

(1) Jordy Kafwe will develop CRAFT; represented by blue in the Gantt chart.

(2) Matthew Craig will develop TempTing; represented by red in the Gantt chart.

References

[1] 2021. Experiences of Using WDumper to Create Topical Subsets from Wikidata. *CEUR Workshop Proceedings* 2873 (2 June 2021).

[2] K. Arrieta, P.R. Fillottrani, and C.M. Keet. 2024. CoSMo: A multilingual modular language for Content Selection Modelling. *ACM/SIGAPP Symposium on Applied Computing (SAC '24)* 39 (2024). https://doi.org/10.1145/3605098.3635889

[3] T. Ball. 2020. *Writing an Interpreter in Go.* Germany. https://interpreterbook.com/

[4] A Gutman. 2022. *Abstract Wikipedia/Template Language for Wikifunctions/Scribunto-based implementation.* https://meta.wikimedia.org/wiki/Abstract_Wikipedia/Template_Language_for_Wikifunctions/Scribunto-based_implementation

[5] A. Gutman and C.M. Keet. 2024. *Abstract Wikipedia/Template Language for Wikifunctions.* https://meta.wikimedia.org/wiki/Abstract_Wikipedia/Template_Language_for_Wikifunctions#Example_templates

[6] R. Harris. 2024. *Svelte Github.* https://github.com/sveltejs/svelte

[7] R. Harris. 2024. *SvelteKit Github.* https://github.com/sveltejs/kit

[8] M. Haverbeke. 2024. *CodeMirror5 Github.* https://github.com/codemirror/codemirror5

[9] G. Hutton. 2016. *Programming in Haskell.* Cambrdige University Press, United Kingdom. https://www.cs.nott.ac.uk/~pszgmh/pih.html

[10] H. Johnston. 2024. *Shadcn-Svelte Github.* https://github.com/huntabyte/shadcn-svelte

[11] M. Klack. 2024. *XYFlow Github.* https://github.com/xyflow/xyflow

[12] K. Lucie-Aiméea, V. Pavlos, and S. Elenac. 2022. Using natural language generation to bootstrap missing Wikipedia articles: A human-centric perspective. *Semantic Web* 13 (2022). https://doi.org/10.3233/SW-210431

[13] Z. Mahlaza and C.M. Keet. 2021. ToCT: A Task Ontology to Manage Complex Templates. In *Joint Ontology Workshops.* https://api.semanticscholar.org/CorpusID:240005311

[14] S. Malyshev, M. Krötzsch, L. González, J. Gonsior, and A. Bielefeldt. 2018. Getting the Most Out of Wikidata: Semantic Technology Usage in Wikipedia's Knowledge Graph. In *The Semantic Web – ISWC 2018.* Springer International Publishing, Cham, 376–394.

[15] L. Martinelli. 2023. *Wikifunctions FAQ.* https://www.wikifunctions.org/wiki/Wikifunctions:FAQ

[16] M. Morshed. 2023. Using Wikidata Lexemes and Items to Generate Text from Abstract Representations. *Semantic Web* (2023). https://www.semantic-web-journal.net/content/using-wikidata-lexemes-and-items-generate-text-abstract-representations-0

[17] S. Ramírez. 2023. *FastAPI Github.* https://github.com/tiangolo/fastapi

[18] D. Vrandecic. 2020. Collaborating on the Sum of All Knowledge Across Languages. (10 2020). https://doi.org/10.7551/mitpress/12366.003.0016 arXiv:https://direct.mit.edu/book/chapter-pdf/2247832/9780262360593_c001200.pdf

[19] D. Vrandečić and M. Krötzsch. 2014. Wikidata: a free collaborative knowledgebase. *Commun. ACM* 57, 10 (sep 2014), 78–85. https://doi.org/10.1145/2629489

[20] D. Vrandečić. 2020. Architecture for a multilingual Wikipedia. (2020). https://doi.org/10.48550/arXiv.2004.04733 arXiv:2004.04733 [cs.CY]

[21] X. Wang, T. Gao, Z. Zhu, Z. Zhang, Z. Liu, J Li, and J. Tang. 2021. KEPLER: A unified model for knowledge embedding and pre-trained language representation. *Transactions of the Association for Computational Linguistics* 9 (2021), 176–194.

[22] A. Wathan. 2024. *Tailwind Github.* https://github.com/tailwindlabs/tailwindcss

[23] D. Zeman. 2024. *Universal Dependencies Github.* https://github.com/UniversalDependencies

Risks

Risk	**Project**	**Category**	**Likely (1-5)**	**Impact (1-5)**	**Mitigation**	**Monitoring**	**Management (Contingencies)**
Unrealistic Schedule	Both	Planning	3	3	Gantt Chart for planning.	Current progress compared against gantt chart.	Additional features can be dropped.
Misunderstanding of Requirements	Both	Planning	1	4	Project plan will be reviewed before execution.	Meetings with the supervisor will be held.	New plans will be drafted.
Scope too Large	Both	Planning	3	3	Scope is detailed in the proposal.	Achievability reassessed after each task is completed.	Additional features can be dropped.
A team member drops out.	Both	People	2	1	Component projects are designed to be independent and parallel.	We will frequently converse to check up on eachother.	Website and Poster will be handled individually.
Libraries are unable to handle expected capabilities	Both	External	3	1	The projects are not overly dependent on any single library.	Documentation will be analysed for features.	New libraries will be used.
Burnout	Both	People	2	5	Work will be done consistently, rather than left to the last stretch.	The current status will be reflected on.	Conversations with peers will help reignite passion.
Unable to establish an adequate visual language to model templates.	Template Tool	Direction	3	4	The grammar of templates will be fully understood before attempting to design a visual language.	The viability of such a language will be assessed during the design stage.	The diagramming tool will be excluded from the project.
Abstract Wikipedia Contributors are unavailable for testing.	Template Tool	People	3	3	Ample time is given to the testing phase.	Early communication will be established with participants. Confirmation will occur closer to the time.	Contributors from other Wikimedia projects will be used. Eg. Wikidata or Wikifunctions
Availability of system reliant on Wikidata	Content Extraction System	External	**1**	**5**	Create backups or find dumps of data	Regular checks to ensure availability of Wikidata.	Switch to a backup source of Wikidata.

Deliverables

CRAFT

Deliverable	**Deadline**
Subset Wikidata	25 June
Setup and Populate local Wikibase	03 July
Query Expansion Algorithm	24 July
API Routes	4 August
Constructor Processing	12 August

TempTing

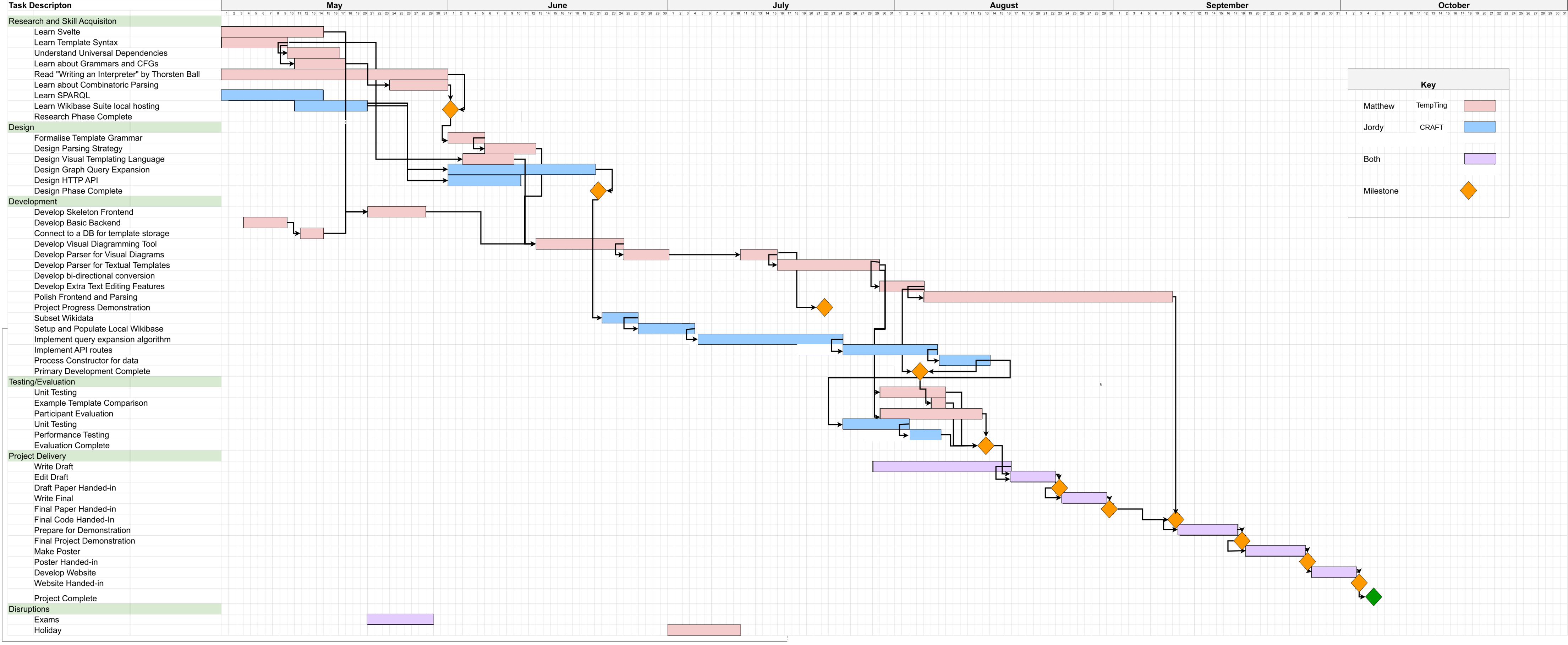
Deliverable	**Deadline**
Basic Backend	09 May
DB Integrated Backend	14 May
Skeleton Frontend	28 May
Visual Diagramming Tool	24 June
Parser for Textual Templates	30 June
Parser for Visual Templates	15 July
Bi-Directional Template Converter	29 July
Textual Template Editor	04 August

Both

Deliverable	**Deadline**
Demonstrable Iteration	22 July
Draft Paper	23 August
Final Paper	30 August
Final Code	09 September
Poster	27 September
Website	03 October

Milestones

Milestone	**Estimated Date**
Research Phase Complete	1 June
Design Phase Complete	21 June
Project Progress Demonstration	22 June
Primary Development Complete	04 August
Evaluation Complete	13 August
Draft Paper Handed In	23 August
Final Paper Handed In	30 August
Final Code Handed In	09 September
Project Demonstrated	18 September
Poster Handed In	27 September
Website Handed In	03 October
Project Complete	04 October


Task Descripton
May
June
July
August
September
October
Research and Skill Acquisiton
Learn Svelte
Learn Template Syntax
Understand Universal Dependencies
Learn about Grammars and CFGs
Read "Writing an Interpreter" by Thorsten Ball
Learn about Combinatoric Parsing
Learn SPARQL
Learn Wikibase Suite local hosting
Research Phase Complete
Design
Formalise Template Grammar
Design Parsing Strategy
Design Visual Templating Language
Design Graph Query Expansion
Design HTTP API
Design Phase Complete
Development
Develop Skeleton Frontend
Develop Basic Backend
Connect to a DB for template storage
Develop Visual Diagramming Tool
Develop Parser for Visual Diagrams
Develop Parser for Textual Templates
Develop bi-directional conversion
Develop Extra Text Editing Features
Polish Frontend and Parsing
Project Progress Demonstration
Subset Wikidata
Setup and Populate Local Wikibase
Implement query expansion algorithm
Implement API routes
Process Constructor for data
Primary Development Complete
Testing/Evaluation
Unit Testing
Example Template Comparison
Participant Evaluation
Unit Testing
Performance Testing
Evaluation Complete
Project Delivery
Write Draft
Edit Draft
Draft Paper Handed-in
Write Final
Final Paper Handed-in
Final Code Handed-In
Prepare for Demonstration
Final Project Demonstration
Make Poster
Poster Handed-in
Develop Website
Website Handed-in
Project Complete
Disruptions
Exams
Holiday
Key
Matthew
TempTing
Jordy
CRAFT
Both
Milestone