Atlas: An Interactive Graph-Based File System for Enhanced Document Relationship Visualization

Daniel Thomas
daniel.cromwel@gmail.com

***Abstract*—Traditional hierarchical file systems often obscure relationships between documents stored in different folders, limiting users' ability to comprehend contextual connections. To address this limitation, this paper presents the development of Atlas, a browser-based application that transforms conventional file systems into interactive graphs. Atlas leverages natural language processing (NLP) techniques to parse documents and generate visual structures linking related files through shared keywords, providing users with a more intuitive way to explore and understand their data. The application was developed using the Scrumban agile methodology, enabling iterative improvements through user feedback obtained from presentations, surveys, and interviews. This approach ensured the design and functionality of Atlas aligned with user needs throughout the product lifecycle. Successfully deployed within an organization, Atlas enhances documentation comprehension and boosts productivity by making relationships between documents readily accessible and visually clear. This innovation highlights the potential for graph-based visualization tools in reimagining file system interactions.**

I. Introduction and Background

The aim of the synoptic project was to create an application that could improve documentation comprehension and discovery in the author's organization. The author designed and developed Atlas, a browser application that can transform a hierarchical file system into an interactive graph of documentation. The graph, named the Atlas Web, displays each document as a single node. Documents are connected if they contain common keywords (Figure 1). The Atlas Web reveals contextual information that could enable the user to better understand the relationships between documents and the concepts they represent. In turn, Atlas could increase the productivity of an organization as the Atlas Web improves understanding and identifies opportunities for collaboration.

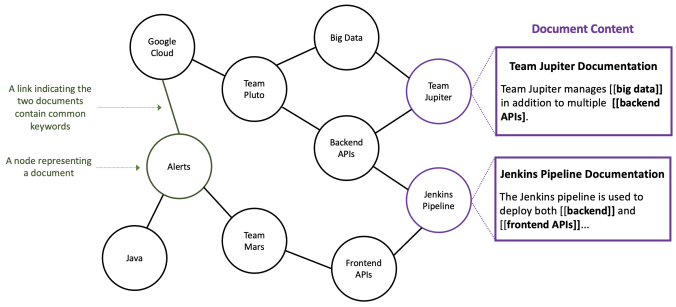

Fig. 1. The Atlas Web is a graph of document nodes that are linked by common keywords.

The idea behind Atlas originated from the author's initial struggle to understand the structure of their organization. The provided directory of documentation was disorganized and difficult to navigate. As a result, it hindered the author's ability to collaborate with different teams in their department. Disorganized file systems are a common problem for large organizations. A recent study reported that 73% of new employees believed poorly maintained document systems hindered their ability to build productive relationships (Rana et al., 2020). External knowledge management tools including Roam (White-Sullivan, 2017) an Notion (Zhao, 2016) have attempted to alleviate these issue by transforming file systems with more intuitive user interfaces. The key purpose of this synoptic project was to develop a cheaper internal solution to improve document comprehension in the author's organization.

Atlas was designed and developed as part of the New Buckinghamshire University Digital and Technology Apprenticeship scheme. Over the course of the synoptic project, the author improved their project management and professional skills by collaborating with members of their organization to create a product tailored to the business's needs. The author also built upon their technical abilities by exploring natural language processing techniques to process and display large amounts of data. In this report the author describes the development process of Atlas and discusses how the project improved their skills.

II. Strategic Rationale and Business Case

A. Business Value

The potential user base of Atlas is inherently wide as file systems are common to nearly all computers. However, the author envisioned Atlas to be particularly beneficial to large organizations such as the authors. Larger organizations are more likely to have more complex documentation systems, which, in turn, require more active management and have a greater chance of becoming disorganized. Atlas could be particularly useful to these organizations, as the Atlas Web can automatically calculate and display the contextual links between documents. These links could passively reveal untapped relationships between separate, uncoordinated areas of the organization.

The Atlas Web is more valuable than even the most organized of hierarchical file systems, as the interface does not limit the user to the context of a folder (Figure 2). Instead, their scope is broadened to include an overview of all documents. The Atlas Web was designed to improve the productivity and efficiency of users, as they are more easily able to understand the relationships between different areas of

the business. Duplicated or disorganized documentation can easily be identified within the Atlas Web, and the resulting clarifications could lead to less duplicated work and more knowledge sharing (Figure 3).

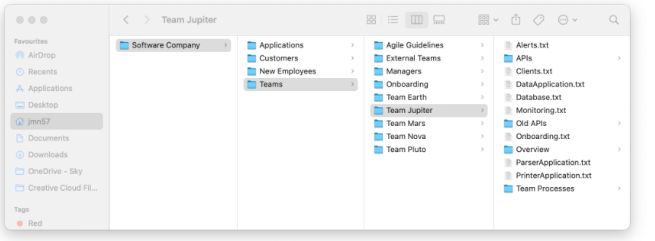

Fig. 2. The traditional hierarchical file system limits users within the scope of folders.

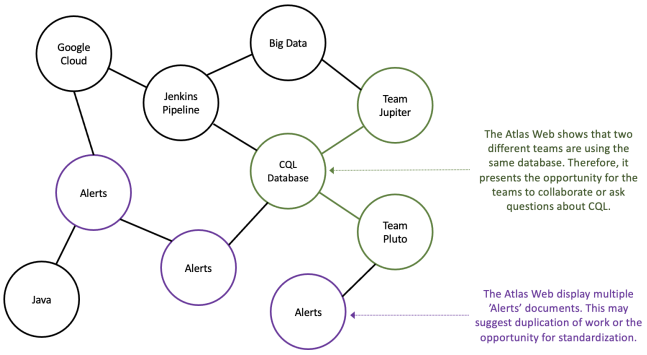

Fig. 3. The Atlas Web creates the opportunity for teams to collaborate and share knowledge.

It is important to recognise that the effectiveness of knowledge management tools is not anecdotal, rather the benefits have a scientific basis. Research has shown visual devices such as mind-maps and diagrams similar to those generated by Atlas are more effective at conveying information than text-based structures (Johnson-Laird, 1983). This preference for visual information has been proven in multiple contexts. For instance, when learning to read, children respond faster when presented with word and picture combinations than when they are presented with written words alone. Although this preference for visual word pairs fades with age, the increased comprehension of diagrammatically displayed information remains in adulthood (Coleman et al., 2018). This was confirmed by a study that revealed students taught with integrated text diagrams performed better on tests as opposed to students who were presented with text (Hegarty et al., 1991).

Visual aids are particularly useful when representing complicated events, concepts, or relationships that would otherwise be difficult to portray with words. Although the degree of comprehension of visual aids will increase with the individuals' cognitive aptitude and profile, generally, the presence of visual aids enables individuals to coordinate both the information from the text and the diagram to create more meaningful linkages (Schieter and Eitel, 2015). This research supports the basis that Atlas could increase the overall comprehension of documentation. Thus, a large organization, such as the author's employer, could benefit from conveying information visually.

Atlas could become particularly effective for new employees who could utilize the Atlas Web to familiarise themselves with the organization's layout. Research suggests that employees may not become effective in their roles until they have a good understanding of the organization's structure (Doherty et al., 2010). Therefore, Atlas could become paramount in reducing the time taken and financial cost of onboarding new members of the organization.

B. Analysis of Commercial Alternatives

To determine whether a knowledge management tool should be purchased rather than developed in-house, the author analyzed competing software solutions. Several software applications offer documentation visualization functionality, the most popular of which is Roam. Roam has been described as a "note-taking application for networked thought" (White-Sullivan, 2017) (Figure 4A). Like Atlas, Roam links documents by keywords to produce an overview of all documentation. Roam gained popularity as an early visual note-taking tool and was recently valued at $200 million. Roam is used by clients such as CodeAcademy, Capgemini, and Match to support and share their large documentation structures with employees (Clark, 2020). In contrast to Roam, Obsidian is a newer, freely available graph-generating tool that is available to download locally (Figure 4B) (Xu, 2020).

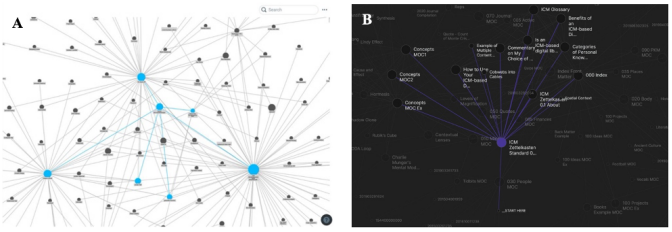

Fig. 4. The graphical interfaces of competing software applications offering visual document structures; Roam (A) and Obsidian (B).

Although current documentation visualization tools offer similar functionality as Atlas, the development of an internal tool would be more beneficial to the author's organization as it would be cheaper, more secure, and available to easily tailor or extend to meet the organization's needs. Roam operates on a subscription-based financial model. For an organization with over 500 members, Roam can cost up to $40 per user annually (Roam, 2021). Such contracts could lock the organization into a costly product, which may turn out not to be an appropriate solution. Cheaper solutions such as Obsidian may be appropriate for smaller organizations whose employees have good technical knowledge. However, for an organization with fewer technical employees, it is unlikely the application would be effectively used as it requires the ability to use command-line tools effectively. On further exploration, the author found the market for freely available knowledge

management tools to be limited by the ability to display large quantities of documentation to non-technical users clearly. The author believed a browser-based solution that could handle a large documentation base would be most appropriate for their organization.

C. Procurement

The author's organization procures new management tools on a monthly basis via an open forum meeting. The author raised the idea of developing a new internal knowledge management tool for the committee, which was met with interest by several members.

Typically, the forum would consider the financial cost of a new product, but as Atlas was developed as part of an apprenticeship degree, they noted that there was no immediate cost to the organization. However, after an initial development period, further development and maintenance would require financial funding. These costs are anticipated to be relatively low as the application would be functionally complete by the end of the initial development period. Furthermore, the cost of hosting the application would be relatively low, as the organization had already negotiated a corporate contract with Google Cloud Platform. As Atlas would not have any user limits or subscription levels, this made this internal solution cheaper than the procurement of popular commercial alternatives.

D. Value to Author

The intention of Atlas was not only to create a useful product for the organization but also to demonstrate and improve the author's technical skills. Atlas presented the opportunity to develop and manage a greenfield project. The author purposely chose back-end technologies familiar to their organization to improve their abilities for use in the workplace. Furthermore, as other employees were consulted for the project, the author was able to build working relationships within the organization.

III. Ethical Considerations

In the development of software, it is vital to ensure permissions are obtained, identities are obscured, and confidentiality is maintained (Winter, 1995). When consumer research was conducted for Atlas, the author ensured that all participants in the research were aware of the intentions of the project. To ensure the confidentiality of those who engaged in any research or testing of Atlas, no person was named in the report. As both the project and report were independent of others' work, no descriptions of other projects were negotiated with any other person.

As Atlas parsed user-inputted files, it was important to ensure that the content was secure. Atlas did not utilize a database; rather, the application was compiled locally, and files were stored in memory on the user's machine. Therefore, there was no concern for the safety of the data as external parties could not access it. To prevent the misuse of the local artefact, the author conducted a testing and security analysis, which resulted in upgrades to all dependencies to versions with no known security flaws.

All libraries utilized by Atlas were published via free and open-source licenses. A list of them can be found in the Atlas GitHub repository.

The ethical checklist required for the synoptic project can be found in Appendix 2.

IV. Aim and Objectives

A. Project Aim

The aim of the synoptic project was to create a browser application that could transform a file system into an interactive graph of documents linked by common keywords. The graph, known as an Atlas Web, would clearly demonstrate the relationships between different documents and, as a result, would improve documentation comprehension and collaboration within the author's organization. The author aimed to design, develop, and deploy the greenfield product within their organization. Atlas could then be used throughout different departments and become maintained by a team of developers.

B. Project Objectives

The author outlined four objectives at the start of the synoptic project that encompassed product design, development, and deployment.

O1. Design a graphical user interface that clearly demonstrates the relationships between documents connected by common keywords.

O2. Develop a minimal viable product that can transform a local file system into an Atlas Web.

O3. Gather and incorporate user feedback on the minimal viable product to develop improved iterations of the application that contain user-requested features.

O4. Deploy Atlas into a production environment so that it can be used by organization members.

V. Risk Management

TABLE I
RISK ASSESSMENT AND MITIGATION STRATEGIES

Id	Description of Risk	Likelihood	Risk Resolution Action	Impact on Project Aim	Impact on Project Objectives	Impact on Project Plan
R1	Underestimate the author's ability to implement the front-end design	High – The author had little experience with front-end development.	Mitigate – The author chose a front-end language they had basic experience with.	The Atlas Web interface may remain incomplete or may not aid document comprehension.	The interface may not depict document relationships clearly (O1). User feedback may not be implemented effectively (O3).	More time allocated to front-end development or a static interface shown in the report.
R2	Underestimate the ability of the chosen technology to efficiently generate the Atlas Web	Moderate – Chosen based on experience and performance metrics.	Avoid – Technologies were tested and substituted if inefficient.	Poor performance may reduce usability and document comprehension.	Atlas Web may not generate efficiently (O2), leading to poor user experience.	More time allocated to technology substitution to ensure efficiency.
R3	Inadequate user engagement	Low – The author secured feedback from 8 users.	Avoid – Prospective users agreed to test the application.	The product may not align with business needs or improve comprehension.	Future versions may lack user feedback (O3), reducing usability.	More time allocated to finding testers, possibly limiting feedback availability.
R4	Inadequate Time Management	Low – The project was scheduled alongside work and study commitments.	Mitigate – The development plan included buffer periods.	Fewer features may be implemented, leading to an incomplete product.	Could delay the development of a minimal viable product (O2) or deployment (O4).	Development stages may be delayed, risking incomplete implementation.
R5	Poor code quality	Moderate – The backend language was familiar, but the front end lacked prior experience.	Mitigate – Research and implementation of coding best practices.	Poor maintainability and debugging could reduce product longevity.	Could hinder minimal viable product (O2) or future development (O3).	Bugs may delay the development timeline and reduce maintainability.
R6	Lack of ownership	Moderate – Plans to demonstrate the application to find a maintenance team.	Mitigate – Regular presentations were conducted.	Lack of ownership could lead to discontinuation.	Without ownership, application maintenance may be neglected (O3).	Without departmental ownership, the product may be retired.

VI. LITERATURE SURVEY

In this literature survey, the author explores the theory, technology, and project management techniques that were used in the design and development of Atlas. In order to meet the outlined aims and objectives, the author explored unfamiliar technical areas such as natural language processing and artificial intelligence. To ensure the appropriate project management style was selected, the author reviewed popular management styles and selected the appropriate method.

A. The Algorithm Behind Atlas

The underlying algorithm that connects related documents in the Atlas Web is based on the Zettelkasten method. The Zettelkasten method was developed by Niklas Luhmann (1927 – 1998) (Ahrens, 2017). Luhmann was a prolific sociologist who championed neither the short nor long-term memory of the human brain could comprehend large quantities of data (Ludecke, 2015). To develop a more intelligible solution to storing and organizing notes, he developed the Zettelkasten method. The Zettelkasten method states a single index card contains one idea, which would be singularly referenced by a sequential number (i.e., 1). If a new note was to be made with the same theme, Luhmann then added a new index card and appended the reference with a letter (i.e., 1a). Then, if another note related to the branched index card, it would then be similarly referenced to (i.e., 1b) (Figure 5). The output of the Zettelkasten method is a database of index cards that can be extended indefinitely in any direction as each index card pertains to a static reference (Forte, 2020).

The Zettelkasten method produces a fluid but organized network of notes. The relationships between notes are clearly displayed, making each note more valuable as ideas are organized into reusable and reflective threads (Gibson, Gregory, and Robson, 2005). The graph of relationships provides a clear external basis of existing knowledge which compliments a more natural thought process as users can focus on elaborating ideas (Gibson, 2020). In turn, this can increase productivity as knowledge is gained more quickly, and users can make more

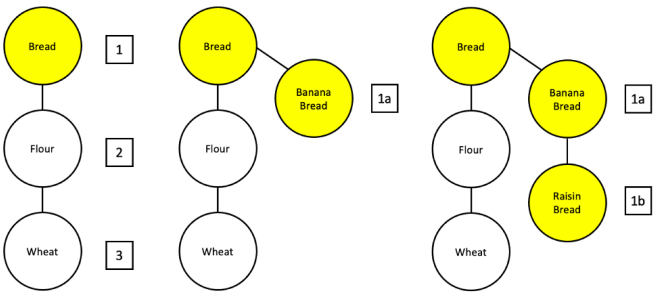

Fig. 5. Niklas Luhmann's Zettelkasten method enables users to create a database of notes linked by topic. This is the foundation of the algorithm used by Atlas.

informed decisions (Glaser and Strauss, 1967). A key benefit of creating an application that uses this algorithm is that the referencing is automated. Therefore, instead of storing and searching for physical cards to reveal information, Atlas will calculate and display the relationship instantly.

A potential criticism of the Zettelkasten method is that the user may find that they are creating ideas that are contradictory or paradoxical. For instance, a user may find that two notes exist, with differing information. However, it could be argued that this is not a limitation of the Zettelkasten method, rather it is a feature. The Zettelkasten method brings these contradictions to light, adding ideas to a discussion. Contradictory data is valuable to discussions, as it represents a misalignment of knowledge and often opens new paths of inquiry (Forte, 2020). By placing incompatible theories side by side, the method encourages objectivity and shares perspectives. This independent thinking counteracts confirmation bias – our tendency to just believe information that we already know – and forces the user to confront misunderstandings (Gibson, Gregory, and Robson, 2005). If Atlas was used in a large corporation, the revelation of contradictory or duplicated notes could lead to discussions or collaboration to align team and departmental knowledge.

B. Project Management Techniques

Software project management techniques are both the science and art of planning and directing software projects (Wasserman, 2010). To be effective, software project management techniques should account for the whole project life, from the initial product design to the planning of tasks, effort estimation, work supervision, and the testing of the end product (Dyba, Dingsoyr, and Moe, 2014). Such management practices have been developed with a culmination of years of trial and error. Although each technique may differ in practice, they all share the ultimate goal of building better software (Hoda, Noble, and Marshall, 2008).

The agile development approach was used to develop Atlas. In agile development, developers work with end-users to implement functionality in iterative cycles. Rather than having a fixed plan, developers respond to changing requirements in real-time (Stellman and Greene, 2008). This ethos reflects the principles outlined by the original Agile Manifesto (Fowler and Highsmith, 2000). The agile methodology opposes the traditional well-defined linear sequence of activities that traditional management techniques such as Waterfall define. Instead, the agile methodology recognizes that activities may require a shorter time between planning and execution. As a result, planned features do not contain all the details of the implementation and are not planned far in advance (Hoda, Noble, and Marshall, 2008).

A particular offshoot of agile known as Scrumban was selected as the appropriate project management technique for Atlas. Scrumban is an amalgamation of two popular agile practices: Scrum and Kanban (Table 2). Both of these practices promote the fundamental agile tenant of utilizing iterative and incremental techniques to quickly gain feedback on development cycles. The Scrum process defines specific individual roles, work requirements, and time boxes that define clear developmental goals. Kanban, on the other hand, offers a system for visualizing a continuous flow of work. Scrumban provides the structure of Scrum and the flexibility and visualization of Kanban.

The development of Atlas was well suited to Scrumban as the project exhibited high variability in requirements, skills, and technologies (Augustine et al., 2005). Therefore, Scrumban would provide a clear progression of the project tasks. Progression awareness was particularly important for Atlas as the project was developed intermittently by a single developer. Furthermore, the short development cycles, common to agile practices, enabled the user to quickly gain user feedback. These different perspectives enabled the quick identification of problem areas and the subsequent prioritization of features (Asra, Sobia, and Khan, 2014).

The author deviated slightly from the pure Scrumban style, as they favored two key Scrum aspects: a shorter iteration cycle of two weeks and the presence of a backlog. The smaller development cycles and backlog promoted more forward planning and helped the author remain organized. In addition to Scrum and Kanban, the author also took up the Extreme Programming practice of utilizing spikes. Spikes are time-boxed explorations into technologies or solutions (Jeffries, 2001). Spikes were appropriate for Atlas as the author was exercising weaker skills - such as front-end development - that required further research before committing to a technical solution.

TABLE II
AN OVERVIEW OF SCRUM, KANBAN, AND SCRUMBAN DEVELOPMENT METHODOLOGIES. HIGHLIGHTED CELLS REFLECT THE CHOSEN MANAGEMENT TECHNIQUES. CONTENT ADAPTED FROM ARBAHAM, 2021.

	Scrum	**Kanban**	**Scrumban**
Timebase	1- 4 weeks	No time base	3 month buckets
Rules	Constrained process	Flexible process	Slightly restricted process
Roles	Product owner, scrum master, scrum team, stakeholder	No specific roles required	No specific roles required
Board	Defined/resets each week	Persistent – the Kanban board	Persistent – the Scrumban board
Prioritization	Backlog	Optional	Recommend on each planning
Estimation	Before sprint starts	Optional	Optional
Planning routine	Sprint planning	On-demand	On demand
Performance metrics	Burn down	Cumulative flow diagram	Average cycle time

C. Natural Language Processing

In this project, the author researched natural language processing techniques. Natural language processing (NLP) is a sub-field of computer science that revolves around analyzing, processing, and representing human language (Damerau and Indurkhya, 2010). Researchers of natural language processing apply machine learning algorithms to speech and text to achieve human-like language processing (Chowdrhy, 2005). The application of NLP research includes predictive typing, speech recognition, and spam detection (Ventsislav, 2018).

The fundamental feature of Atlas is the ability to transform a hierarchical file system structure into the Atlas Web. To achieve this, all folders and documents in the file system must be scanned, and keywords must be identified and stored. In the field of NLP, this process is known as deductive parsing (Trujillo, 2018). A deductive parsing algorithm has two distinct elements: a parser, which will scan each word, and a tokenizer, which will check for pre-defined tokens in the input (Chowdrhy, 2005). For Atlas, the keywords identified by the tokenizer were surrounded by square brackets (e.g., [[keyword]]).

In addition to parsing and tokenization, Atlas would also be developed to utilize the natural language processing techniques: stemming and lemmatisation. The goal of both stemming and lemmatization is to reduce inflectional forms of words in order to form a common base (Jivani, 2011). These processes would be integral to Atlas, as they would link multiple derivatives of the same word in the Atlas Web. Understanding the subtle difference between these two processes was important in order to prioritize and implement the techniques (Figure 6). Stemming is the crude process of removing the end of words in order to chop off derivational affixes (i.e., 'run' and 'running'). Lemminisation is a more complex process that conducts a morphological analysis of words, removing inflectional suffixes to return the dictionary for a word known as a lemma (i.e., 'study' and 'studies') (Ventsislav, 2018). This research led the author to prioritize the implementation of stemming first, with the goal of implementing the more complex immunisation for the end product.

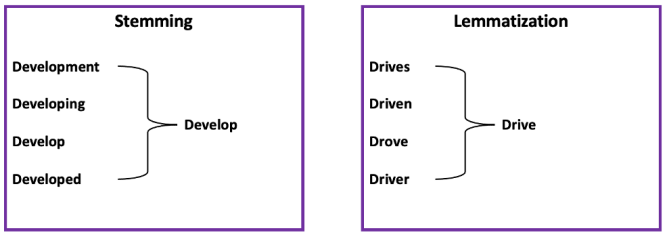

Fig. 6. The difference between two common natural language processing techniques: stemming and lemminisation.

D. Artificial Intelligence

Atlas would incorporate elements of artificial intelligence as it would be able to recommend common keyword tags. Artificial intelligence enables machines to complete tasks that would require intelligence if completed by a human (Boden, 1997). A famous thought experiment that encompasses artificial intelligence was demonstrated by Alan Turings' Imitation Game, a computer program that displayed intelligence by deceiving a human interrogator into believing they were communicating with a human (Turing, 1950). Another example in the field of artificial intelligence expanded from rule-based systems with the advancement of neural networks. Neural networks are based on a series of algorithms loosely modelled by the human brain that interpret patterns from raw data.

The implementation of artificial intelligence into Atlas will improve upon the underlying Zettelkasten algorithm by automatically detecting common, non-tokenized keywords and suggesting them to create new links between documents. This tag suggestion feature will parse all of the files and identify commonly used words using rule-based logic. Importantly, the application must identify and omit words such as 'the,' 'a, and 'and' to prevent the recommendation of useless tags. This functionality will demonstrate a level of intelligence, as the application automates a task that would take a human significantly more effort to complete.

E. Language Choice

Upon embarking on a greenfield software project, it is important to ensure the appropriate technology is chosen to meet the functional and non-functional requirements. The development of Atlas was divided into two parts: the front end and the back end. The back-end performed the transformation of documentation into an object that represented the Atlas Web. Scala was selected as the back-end language as it was familiar to the author and highly performant. Scala is both a functional and object-orientated high-level language that runs on the JVM. It has access to the ecosystem of Java libraries, and it is, on average, 20% more performant than Java (Roper, 2012). Scala libraries such as Akka are designed to utilize resources efficiently by leveraging actor systems asynchronous thread management, which in turn makes applications highly scalable and responsive (Roestenburg et al., 2016). As Scala is statically typed, it is able to check for common classes of errors at compile time, unlike other common back-end languages such as Python (Kerringham and Ritchie, 1978). Although other back-end languages could have been used to build Atlas, the author was keen to improve and demonstrate their skill with Scala as they are utilized in their day job.

The front end of Atlas rendered the Atlas Web. The author had little experience with front-end development, so they selected React Typescript as their front-end framework. React is a library for building user interface components. It is the most popular front-end library and has a strong online community, which offers a large amount of beginner support online (Pitaliya, 2021). The author selected React as they had a small amount of prior experience with it and felt confident they could create the Atlas with the online resources available. As the Atlas web could be divided into a few repeated components, the author was confident React's reusable component system could create the interface with minimal additional complexity

to the code. The author decided to choose React Typescript, the superset of JavaScript, as it adds static types to the code, which in turn reduces the chance of runtime errors (Finley, 2012). Other front-end options, such as Swift or Angular, could have been performant enough for Atlas, but they were not selected as they would require more time for the author to develop due to their lack of experience.

VII. Methodology

A. Timeline

Although the development of Atlas utilized an agile project management approach, the author created an initial timeline of the development to ensure that the product could be completed before the deadline (Figure **??**). The timeline was flexible to allow for any unforeseen problems and included key milestones for gathering feedback. The author purposefully ensured that the minimal viable product was created early in the development process so they could quickly gain user feedback. Feedback was collected in different formats, including a survey, focus groups, and an interview. The author planned to create three incremental versions of the application, each one including improved and additional features. The timeline also scheduled regular discussions with the project supervisor and the author's manager to ensure the project aligned with the remit and progressed the developer's skill-set.

B. Scrumban Approach

The discussion in the literature survey concluded that the Scrumban approach would be the appropriate project management style for the development of Atlas. Scrumban is a subset of agile development that uses iterative development cycles that perpetuate adaptive responses to user feedback (Stellman and Greene, 2008). The author tracked the work completed in each iteration on a GitHub KanBan board (Figure 7). The Kanban board enabled the author to create tickets with detailed acceptance criteria (Figure **??**). The author decided not to explicitly estimate the tickets as there was little value to be gained as they were the sole developers and had nobody to coordinate with. Instead, they focused on creating tickets with less than 4 hours of work to ensure that they did not remain on a single ticket for a long period of time. Furthermore, the author did not abide to strict sprint timelines, instead, they aimed to spend no more than 6 weeks on each iteration of Atlas, and added to the backlog of tickets when necessary. Although there are more detailed and trackable Kanban board solutions, GitHub was selected as it provided the required features and was conveniently hosted within the code repository.

C. Incorporating User Feedback

The incorporation of user feedback is a key tenant of agile development (Stellman and Greene, 2008). Both oral and written feedback were captured during the development of Atlas. Once the minimal viable product was complete, the author showcased the application and collected feedback in a survey. The survey captured mainly quantitative data, as closed

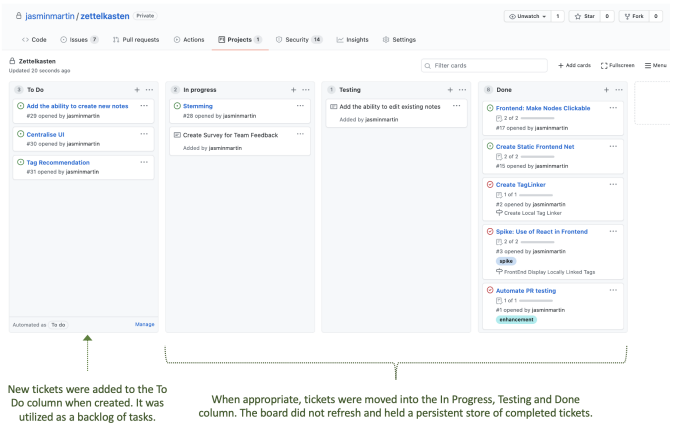

Fig. 7. The GitHub Kanban board is used to track upcoming and completed tickets.

multiple choice questions prompted the respondent to rate the demoed user experience. Qualitative feedback was captured, too, in a series of open questions, which enabled the user to give more detailed opinions on the current and upcoming features. Both types of data were analyzed to provide the author with an idea of the user's needs and preferences. The results of the survey influenced the features incorporated into the second version of the application and resulted in the refinement of the features included in the minimal viable product.

Further feedback was captured in a focus group and an interview. As these two types of data collection required real-time feedback, they prompted more honest and detailed responses than the initial survey (Dawson, 2015). These demonstrations enabled the users to interact with Atlas, which in turn identified difficulties with the user experience. Live data collection was particularly important for the development of Atlas, as only a single developer created and designed the product. Incorporating other perspectives and viewpoints on the functionality made the product more valuable to a larger audience. The author decided to schedule a focus group for the end of version 2 development and later an interview for version 3, as the interview would elicit more in-depth and opinionated reviews (Cottrell, 2014). The notes collected on opinions, questions, and ideas were analyzed to determine any recurring themes that were addressed in the later versions of Atlas.

VIII. Requirements and Design

Here, the author describes the requirements solicitation process that shaped the design of Atlas. The initial product idea stemmed from the author's personal experience of struggling to comprehend a large, complex, hierarchical file system of documentation. As a result, the initial requirements were largely formed from the author's desire to demonstrate the benefits of generating the Atlas Web structure. Later iterations of Atlas incorporated user feedback - following the agile ethos of continuously tailoring the product to the user needs

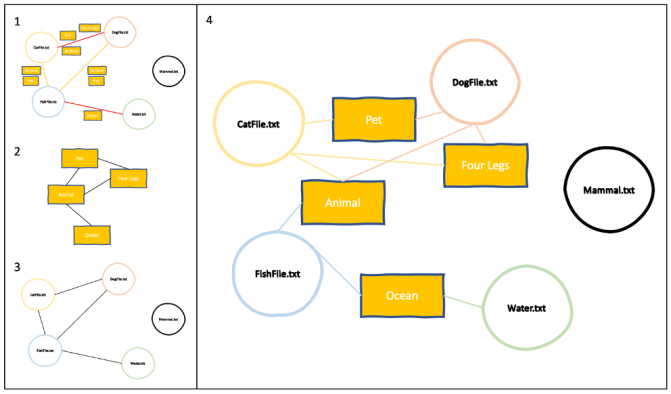

Fig. 8. The development of the initial wireframe of the Atlas web demonstrated the connections between documents with common keywords. The author experimented with different designs before settling on the circled document, boxed keyword arrangement (4).

TABLE III
THE FUNCTIONAL REQUIREMENTS AND USE CASES THAT DEFINED THE MINIMUM VIABLE PRODUCT.

Phase	Id	Functional Requirement	Use Case	Use Case Description
MVP	F1	To form relationships between tagged words	Calculate keyword links	When the system is provided with a file system directory, it must parse the file contents and calculate the Atlas Web.
MVP	F2	To display relationships on the user interface	Display Atlas web	A user should be able to view the Atlas Web that displays all of the files and the relationships between them.

(Stellman and Greene, 2008). In this section, the author details how user feedback influenced each iteration of the product.

A. Iteration 1: MVP Concept

The initial functionality of Atlas was demonstrated in the minimal viable product (MVP). An MVP is a product with enough features to attract customers to validate the base functionality. The key purpose of an MVP is to enable the maximum amount of user feedback with the least amount of effort (Ries, 2009). The MVP was targeted toward adults with low technical experience. This demographic was selected as research into competitor products revealed that popular knowledge management tools expected a moderate degree of technical literacy. As a result, the author purposefully limited the number of steps in the user journey required to generate the Atlas Web (Figure **??**).

The author recognized the key features of Atlas with the aid of product wireframes (Figure 8). The original design of the Atlas web connected documents with common keywords; and displayed both the keyword and the name of the document in the Atlas web. The Atlas Web was designed to clearly depict keywords in orange boxes and documents in multi-colored circles. From this visual depiction, the author recognized the two key functional requirements to be the ability to form relationships between documents and keywords (F1) and the ability to display the Atlas web on the user interface (F2) (Table 3).

Before developing Atlas, the author explored the requirements in greater detail with the aid of use cases and use case descriptions. Use case descriptions illustrate the behavior of a system without specifying how they are implemented (Hoda et al., 2008). On each iteration of the application, the author defined the use case descriptions for new functional requirements to help envision the technical implementation of the new features (Tables 3-6).

The author successfully implemented the initial functional requirements for the MVP (Figure 9). Note, on seeing the user interface, they decided to use circles instead of squares to make the graph appear less complex.

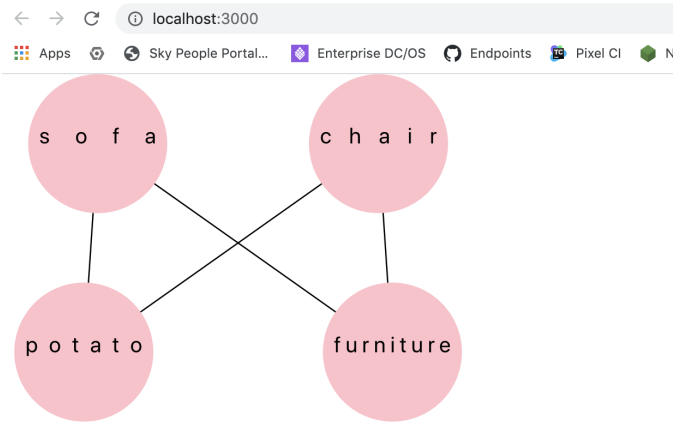

Fig. 9. The user interface of the Atlas MVP demonstrated the application's ability to connect documents based on common tokenized keywords.

B. Iteration 2: Showcase and Survey

To formulate the requirements used to iterate upon the MVP, the author showcased Atlas to members of their department. The presentation described the project aims and included a demonstration of the MVP Atlas Web (Figure 10). The dual purpose of the presentation was to inform colleagues of the author's apprenticeship work and to collect feedback on the MVP. The presentation was followed by a survey that would inform the author of the success of implemented requirements and influence the development and prioritization of future requirements. Due to COVID-19 restrictions, the showcase was held on Microsoft Teams with an audience of 15 members of the author's organization.

The author purposely chose to showcase the MVP to members of the department early in the development process to generate an awareness of the project to the department. The author hoped this could lead to opportunities to collaborate with members of the department and could help find a team to maintain the application on completion. A stratified sample of 25 members of the department were invited to the showcase, but unfortunately, 10 members could not attend. The demographic of the audience was largely male (80%) and consisted of employees in technical roles. The author was aware that this population may not be reflective of the final user base and could show a bias toward both male and technical opinions. This limitation was reflective of their department's population. However, the author noted if the product was to be utilized

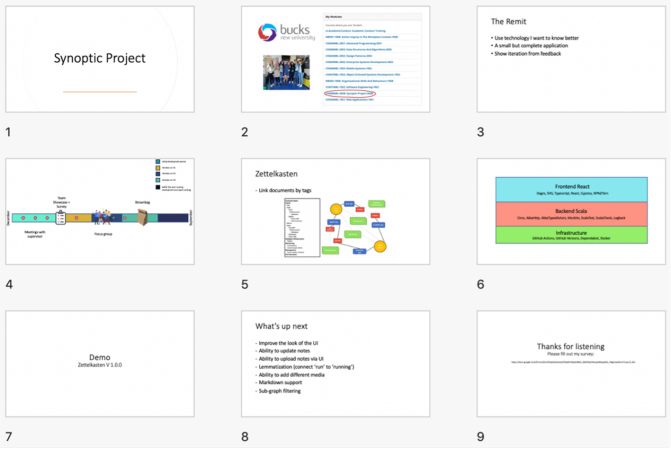

Fig. 10. The slideshow presentation showcased the project background and the MVP to prospective users.

in the organization, this would likely be the demographic that utilized the product.

After the showcase, attendees were asked to complete a survey (Appendix 1). The survey offered a low-cost and convenient mechanism for generating a large amount of data (Jones et al., 2013). It had two key purposes – the first was to understand whether the audience had understood the purpose of the current MVP concept, and the second was to give the opportunity to suggest future features. The survey contained primarily closed questions used to collect quantitative data on topics such as the user interface, the user journey, and the user's desire to use the application. Open questions enabled the respondents to suggest new features that the author may not have previously considered. The data was compiled and analyzed to influence the priority of the next functional requirements (Figure 11).

The data analysis revealed that the current MVP concept was well understood, as the average audience member rated product comprehension at 7.9, user interface comprehension at 9.1, and user journey comprehension at 8.8. The audiences rated their likeliness to use the product as 7.1. Comments from the survey suggested this lower value may stem from concerns that the Atlas Web may not be able to handle larger document structures than demonstrated. The author ensured that Atlas Web would be demonstrated with more documents in the future. Comments on the current MVP design also highlighted the fact that Atlas Web was not centralized.

The survey prompted users to vote for Atlas's upcoming features. By far, the most voted-for new feature was the ability to open the documents within the application (F3). The second, third, and fourth most-voted features were the ability to add (F4), delete (F5), and edit (F6) documents. These formed the next four functional requirements for Atlas (Table 4).

Interestingly, three survey respondents suggested the same additional feature – the ability to zoom in on the Atlas web interface. This ability proved more popular than the author's suggested sub-graph feature. Although this feature

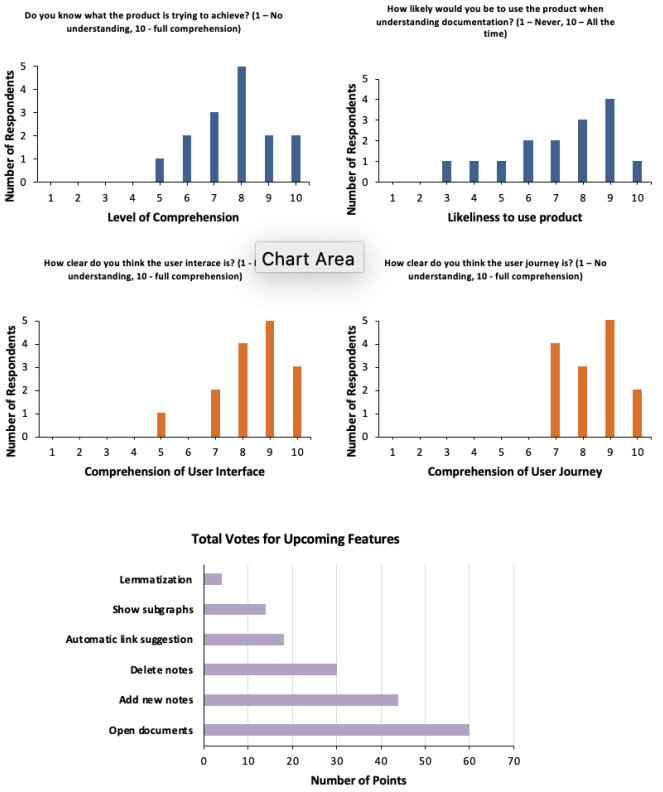

Fig. 11. The minimal viable product survey revealed users' comprehension of the current product and future features that they would like to see implemented.

TABLE IV
THE FUNCTIONAL REQUIREMENTS AND USE CASES THAT DEFINED THE SECOND ITERATION OF ATLAS.

Phase	Id	Functional Requirement	Use Case	Use Case Description
It. 2	F3	To view document content	View document content	A user should be able to click on an Atlas web document to open a modal which will display the document contents.
It. 2	F4	To add a new document	Add a document	A user should be able to add a new document to the Atlas Web.
It. 2	F5	To delete an existing document	Delete a document	A user should be able to delete an existing document from the Atlas Web.
It. 2	F6	To edit document content in the Atlas Web	Edit a document	A user should be able to edit a document to update the content.

was not incorporated into the second iteration of Atlas due to workload, the author highlighted this as a potential feature in the upcoming focus group.

To aid the development of the second iteration, the author introduced a basic user actor to the system (Figure 12). Atlas only catered to a single actor, as the primary purpose of the application was to automatically transform the file system into the Atlas web. As there was no concept of an admin user, all of the functionalities were carried out by the basic user.

C. Modal and Atlas Web Design

Before the implementation of iteration 2 of Atlas, the author created wireframes to show the modal layout that will display document content. The wireframe provided a visual guide for the developer that ensured the element positions were both appealing and clear (Stone et al., 2005). The author also used the wireframes to update the design of the Atlas Web based on feedback from the survey.

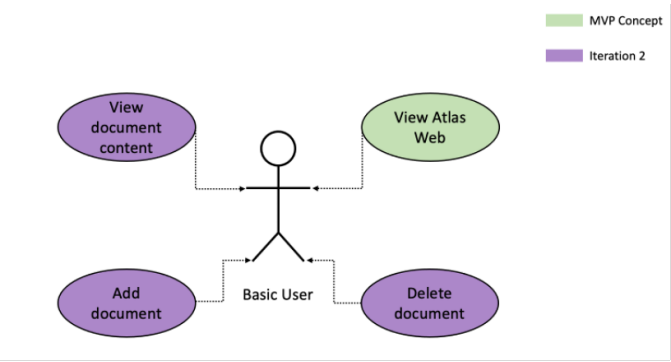

Fig. 12. Actor diagram showing the new functionalities carried out by the user in the second iteration of Atlas.

The Atlas Web represented both documents and keywords as circles. As circles are universally known as clickable buttons, they enticed users to interact with them. Relationships between the files were represented by lines. Lines were chosen over arrows so as not to overcrowd the screen or make the system appear overly complex by adding directionality to the relationships (Najjar, 1998). A user can add a new document by pressing the New File button on the Atlas Web user interface (Figure 13A).

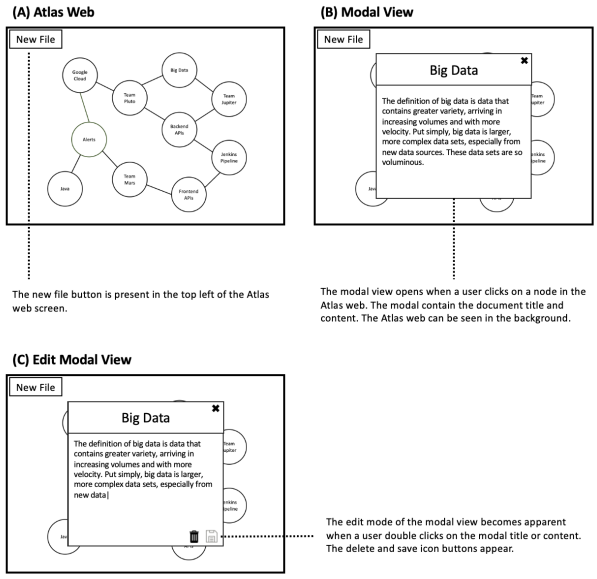

Fig. 13. Wireframes representing the Atlas Web (A), the modal view (B), the edit modal view (C).

When a user clicks the circular document button, a modal appears with the document contents (Figure 13B). The appearance of a modal was chosen rather than a new page, as it provided a smoother transition. The user could easily click on the background Atlas Web or the closing 'X' to close the modal. Therefore, the user should feel confident navigating the product and should not feel lost between windows. Furthermore, the modal was chosen as it provided a distinct experience from the traditional file system, which opens different files in different application windows.

The author decided to keep the design of the application simple and enable the user to both edit and delete the document within the modal view (Figure 13C). To edit a document, the user clicks the body or title of the modal. This mobile-centric approach was taken as it reduced the number of buttons on the user interface. The save and delete icons enable the user to update the document accordingly.

The author completed the development of the modal and managed to produce a design similar to the intended wireframes (Figure **??**).

D. Iteration 3: Focus Group

Once the second iteration of Atlas had been developed, further requirements were gathered with feedback from a focus group. The aim of the focus group was to gather different feelings and perspectives on the product (Gibbs, 1997). The focus group consisted of 6 members of the organization and included a small introduction to the product, followed by a demonstration that enabled users to interact with Atlas. Each user was encouraged to interact with the application so that they experienced all elements of the user journey. The author made note of any comments or concerns with the user experience (Appendix 1). To ensure the focus group was reflective of the user base, a stratified sample of users across the organization was selected to incorporate a broader career range and a balanced gender ratio.

The resulting qualitative data was analyzed to find common sticking points with the application. Three members of the group initially struggled to understand how the application translated the traditional file system into the Atlas Web. It was suggested an instruction guide could alleviate this issue (F7). Furthermore, the focus group reported it was difficult to comprehend the Atlas Web when it had more than 15 nodes. The group responded positively to the author's suggestion of a zoom feature to help alleviate this issue (F8). Finally, on member of the group asked for the ability for Atlas to automatically recommend new connections if documents contained similar content (F9). The author prioritized these three new functional requirements by the ease of implementation as they were aware of time constraints (Table 5).

TABLE V
THE FUNCTIONAL REQUIREMENTS AND USE CASE DESCRIPTIONS THAT DEFINED THE THIRD ITERATION OF ATLAS.

Phase	Id	Functional Requirement	Use Case	Use Case Description
It. 3	F7	To display an instruction guide	Display instructions	A user should be able to see the instruction guide when they first open the application.
It. 3	F8	To zoom into the Atlas Web	Zoom into Atlas web	A user should be able to zoom into different parts of the Atlas Web.
It. 3	F9	To automatically suggest new connections	Suggest new document relationships	The system must be able to suggest potential connections based on common words.

An unexpected development that stemmed from the focus group was the change of the product's name. Originally, the product had been named 'Zettelkasten', after the author of the underlying transformation algorithm. The focus group reported the name had complex connotations and did not describe the product's purpose well. For the fourth iteration of development, the author renamed the product to Atlas, as

it was simpler and better aligned with the product purpose – the creation of knowledge maps.

The author successfully implemented the instruction guide (Figure 14) and the zoom functionality (Figure **??**) and presented these features in the next iteration of development.

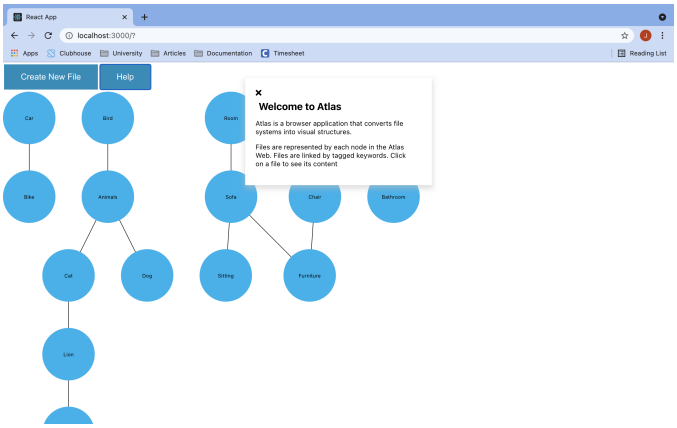

Fig. 14. By pressing the Help button, a user can open the instruction manual to instruct them on how to use Atlas.

E. Iteration 4: Interviews

The author had planned to conduct interviews with several members of the organization to gather more feedback and form more functional requirements. Interviews are particularly advantageous as the author is able to adapt their line of query to the interviewee's responses (Gibbs, 1997). Unfortunately, the author only had time to informally interview one colleague, due to time restrictions. The feedback from the interview was largely positive. The interviewee suggested one additional feature - the ability to connect documents with similar names, such as 'database' and 'databases' (Table 6). This process is known as stemming and has been previously researched by the author. The author successfully implemented the stemmer (Figure 15).

TABLE VI
THE FUNCTIONAL REQUIREMENT AND USE CASE DESCRIPTION THAT DEFINED THE FOURTH ITERATION OF ATLAS.

Phase	Id	Functional Requirement	Use Case	Use Case Description
It. 4	F10	To connect documents with the same stem	Connect documents with the same file name stems	The system must connect documents with the same file name stems.

In order to implement the stemming requirement (F10), the author used a sequence diagram to guide the implementation (Figure 16). Sequence diagrams demonstrate the interaction between systems over time (Fowler, 2003). This enabled the author to understand that when a user-provided the location of a file system, the system could then identify the nodes and edges of the documents by parsing each file individually. The nodes would then be stemmed when they were identified by the system.

F. Class Diagram with Entity Relationships

To translate the use case descriptions into a technical design, the author consolidated class diagrams with the entity

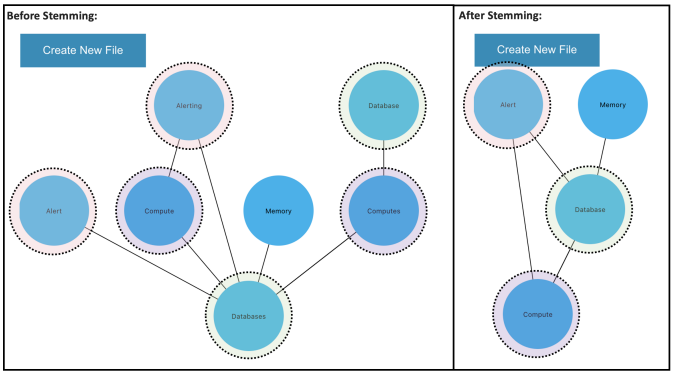

Fig. 15. The author successfully implemented a stemmer to connect files with the same stem.

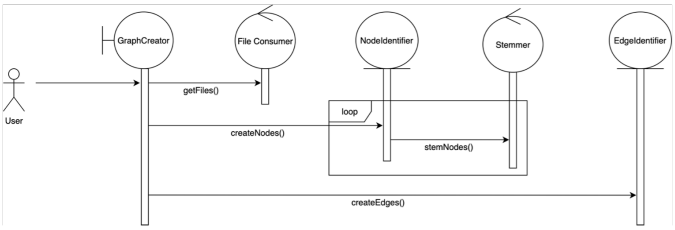

Fig. 16. The sequence diagram demonstrates how a user interacts with the system to create an Atlas web.

relationships schemas to demonstrate the relationship between the classes and objects within the system (Figure 17). The author first created this diagram before the MVP concept was implemented and extended it with each iteration of the application. The diagram was then used as a reference point to guide the development of Atlas.

The case class diagram shows two key packages – the file consumer package consisting of the FileConsumer trait and LocalFileConsumer class, and the GraphGenerator package consisting of eight objects and classes connected to the GraphCreator class. The GraphCreator class utilized the EdgeIdentifier and NodeIdentifier objects to produce the list of files and related nodes, which, in turn, generated the Graph displayed to the user. These objects were supported by the FilesAndTags class, with the resulting objects being combined by the GraphCreator. The NodeIdentifier object was further supported by the Stemmer object, which enabled the stemming functionality.

G. Non-Functional Requirements

Whilst the functional requirements defined the behavior of the application, the non-functional requirements described the system-level attributes such as performance, security, usability, and reliability (Augustine et al., 2005). Non-functional requirements are important as they can determine whether the user has a positive user experience. The non-functional requirements of Atlas were solicited throughout the development process (Table 7). The focus group elicited the importance of the user's ability to understand and comprehend the application features (NF1). The second and third non-functional requirements

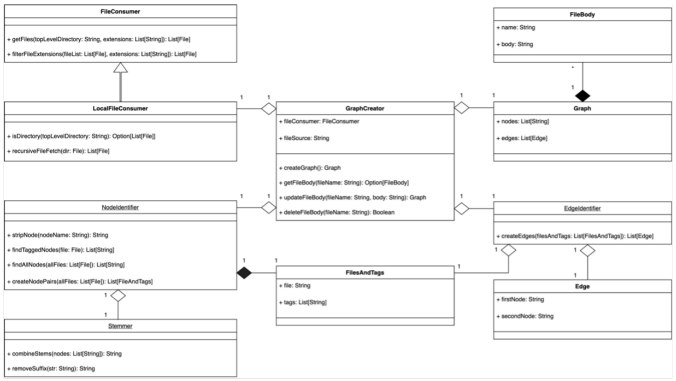

Fig. 17. The combined case class and entity relationship diagram that guided the development of the Atlas system.

concerned with the security (NF2) and speed (NF3) of the application and were influenced by the guidelines of similar applications (Addison-Wesley, 2003). The final non-functional requirement focused on the developer experience (NF4). It was important to maintain a clean code base, as the application may have to be maintained by other members of the department.

TABLE VII
THE NON-FUNCTIONAL REQUIREMENTS OF ATLAS WERE SOLICITED TO PROVIDE A SMOOTH USER AND DEVELOPER EXPERIENCE.

Id.	Non-Functional Requirement	Example
NF1	A user must be able to understand the Atlas Web and be able to navigate the user interface with ease.	A user should be able to understand the application by reading the instructions.
NF2	The security of a system must keep user information confidential and prevent any unwanted outside interaction with the system.	User data must only be stored locally.
NF3	The application performance must respond rapidly to user requests.	A user should be able to see a newly added node appear on the user interface within 500ms^{-1}.
NF4	The application codebase must be well maintained and accessible for future developers.	A linter will be used to reduce the amount of errors and stylistic differences.

IX. DEVELOPMENT

A. Time Management

A total of four iterations of Atlas were developed. The first iteration, the MVP, was completed as planned in early March after 14 weeks of development. The three later iterations of the application did not meet the initial schedule, as the first iteration overran by 3 weeks (Figure 18). Time management problems were highlighted as a low risk in the initial product plan (R4) as a 20% buffer had been added to all time estimates. Although all four iterations of the product were implemented, the author had significantly less time to write the project report than originally allocated.

The author had dedicated half a day per week to the project to allow time for work on parallel university modules. This assumption proved inaccurate, as the high time demands of the other modules resulted in little time for project development. Due to high work demands outside of the apprenticeship, the product could not be developed as part of the author's daily work. As a result, the author found themselves developing the project outside of the allocated hours, which in turn made it difficult to gain momentum on each iteration. Furthermore, the additional task of writing the report led to later stages

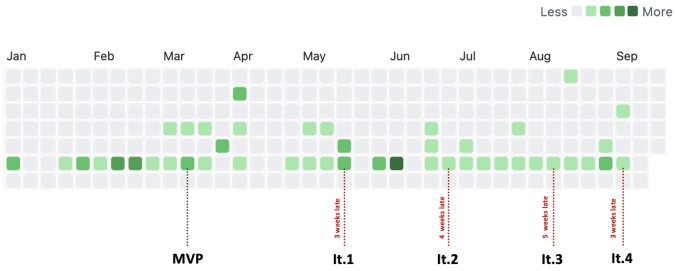

Fig. 18. Annotated GitHub commit chart demonstrates the successful completion of the MVP on schedule, followed by time slippage on the later iterations of Atlas.

of development becoming particularly rushed. Fortunately, the Scrumban agile methodology enabled the author to prioritize important tasks and omit those not required to create a functionally complete product.

B. Completed Functionality

The author implemented 9 of the 10 functional requirements outlined in the original requirements and design section (Table 8). The automatic tag suggestion requirement (F9) was omitted due to the aforementioned time constraints. The author struggled to develop a solution for automatic tag generation that could efficiently store and identify words common to multiple documents as the data set scales. If this requirement was further developed, the author would research Scala frameworks that could store and manipulate large quantities of data, such as the Play Framework (PlayFramework, 2021).

TABLE VIII
A LIST OF THE FUNCTIONAL REQUIREMENTS INCLUDED IN THE FINAL ITERATION OF ATLAS.

Id	Functional Requirement	Completed?
F1	To form relationships between tagged words	✓
F2	To display relationships on the user interface	✓
F3	To view document content	✓
F4	To add a new document	✓
F5	To delete an existing document	✓
F6	To edit an existing document	✓
F7	To display the instruction guide	✓
F8	To zoom into the Atlas Web	✓
F9	To automatically suggest new connections	
F10	To stem similar words	✓

C. Completed Objectives

The author implemented 3 of the four original objectives (Table 9). The author was satisfied that they had created a well-designed product that could efficiently aid users with the transformation of a file system into an Atlas Web (O2). Furthermore, they believe that with the implementation of user feedback, they developed a product that clearly depicts the relationships between documents (O3 and O1). Although the code for Atlas is available on GitHub, the author did not deploy Atlas into a production environment due to the author not being able to have the resources to deploy the application into their department (O4). The author is waiting to see if Atlas will be maintained by the department.

TABLE IX
A LIST OF THE OBJECTIVES INCLUDED IN THE FINAL ITERATION OF ATLAS.

Id	Objective	Completed?
O1	Design a user interface that clearly depicts the relationships between documentation.	✓
O2	Develop a minimal viable product that can transform a specified file system into a web interface.	✓
O3	Gather and incorporate user feedback on the minimal viable product and develop future iterations of the application.	✓
O4	Deploy Atlas into a production environment so that it can be used by organization members.	

D. Spike: Front-end React Development

During the development of the MVP, the author dedicated a day of development to a spike in front-end libraries. A spike is a method for evaluating the impact of new technology that enables the developer to become more confident with the desired approach (Agile Learning Labs, 2021). The author had highlighted their lack of experience with front-end development as a risk from the outset (R1), and as a result, they wanted to explore different solutions for the visualization of the Atlas Web. This exploration would also help the author minimize another risk, the inability of the chosen technology to efficiently generate the Atlas web (R2).

The author explored two key solutions for front-end web generation. The first was to create the Atlas Web with their own React components. From utilizing online resources, the author learned that it was easy to create basic shape components in React, such as the lines and circles which form the basis for the web. The author found shape manipulation in space to be difficult to calculate as the relative coordinates changed with screen size and scale (Figure 19A). The second solution the author explored was the use of the Dagre library. Dagre is a JavaScript library that configures the position of elements in a graph (Pettitt et al., 2013). Using Dagre, the author could easily manipulate the Atlas Web layout (Figure 19B). The author decided to implement Dagre as it provided an efficient bespoke layout engine for generating the Atlas Web. Choosing Dagre mitigated the risk of building a bloated code base as it encompassed the required positional calculations (R5).

E. Technical Challenge: Porter Stemmer Algorithm Implementation

In order to link nodes with the same stem in the Atlas Web, the author incorporated the Porter algorithm. The Porter algorithm removes a suffix from a word to obtain the stem (Porter, 1980). For example, words such as 'makes', 'make' and 'making' will be reduced to 'make' after stemming. The Porter algorithm enabled different inflections of the same word become linked in the Atlas Web.

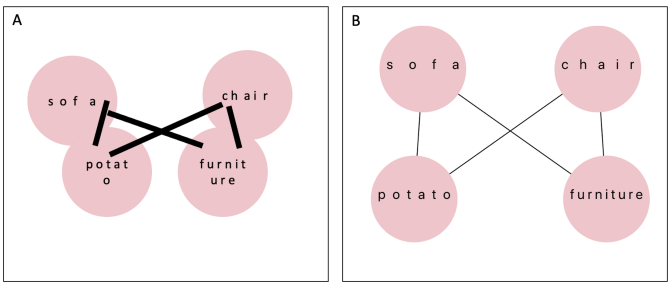

Fig. 19. The result of the front-end spike encouraged the author to use the front-end Dagre library (B) rather than create their own framework (A).

There are three predominant stemming algorithms found in natural language processing: the Porter algorithm (Porter, 1980), the Snowball algorithm (Porter, 2001), and the Lancaster algorithm (Paice, 1990). The Porter algorithm is less complex than the Snowball and Lancaster algorithms, and often produces more intuitive representations of stemmed words. Although, the Porter algorithm is slightly less performant than the Lancaster algorithm (Porter, 2001), as the author saw no latency increases in Atlas Web generation, they chose the Porter implementation. Note, this algorithm is appropriate for other languages, enabling this feature to be used in the author's global organisation (Porter, 1980).

The implementation of the Porter algorithm incorporated rules such as the pattern of consonants and vowels, and the letters in the stem (Figure **??**). The algorithm was particularly challenging, as the author had to manipulate the characters in the file name to calculate the measure of the consonants and vowels, in addition to identifying any rule-triggering suffixes. There were no Scala examples of the Porter algorithm online, therefore, the author was challenged with developing a unique implementation.

F. Technical Challenge: CORS Handler Implementation

During the local development of the MVP, the author was confronted with a communication issue between the front-end and the back-end of Atlas. Atlas was designed to be segregated to simplify the development between the two different areas and to enable respective scaling. However, the communication between the two was initially blocked by the presence of a Cross-Origin Resource Sharing (CORS) Policy error. CORS is a security mechanism in modern browsers that controls the access of resources from outside of the browser domain (Mozilla, 2021). When the author created the front-end server to invoke the back-end CRUD requests, they did not realize that CORS would invoke an OPTIONS security request before sending the intended CRUD call. As the back-end was not configured to handle the OPTIONS request, it resulted in a CORS policy error (Figure 20).

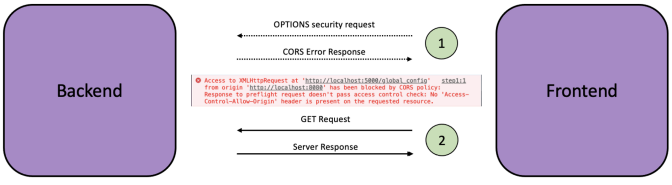

Fig. 20. The author faced the technical issue of a CORS error as the backend was not configured to handle OPTION requests.

To solve the CORS policy error, the author implemented a CORS Handler trait to wrap the back-end server. This class enabled the OPTION requests to the back end and added the correct response headers to validate the security

request. The author had not anticipated the risk of front-end to back-end connection issues as they previously had only interacted with front-end systems in environments with existing solutions to this problem. This error provided the author with the opportunity to learn about inbuilt browser security mechanisms.

G. Technical Challenge: Generating Test Data

In order to test the ability of the back-end to generate the Atlas Web from a file system, the author created a series of unit tests that parsed a local file system. As more tests were added to the project, the test results became inconsistent. This was because certain tests invoked PUT requests to update document content, which, in turn, caused subsequent tests to fail due to incorrect assertions about the document content. To make the tests stateless, the author created a utility class that could create a test file structure. The utility class was invoked before each unit test was run. The re-creation of the file structure ensured that each test had a consistent environment, which previous tests had not manipulated. This also enabled the author to omit the test file structure from the repository, which reduced the project complexity. Thus, the author learned the importance of creating clean, stateless unit tests.

H. Methodology Adaptation: Balancing Scrumban Roles

Overall, the Scrumban methodology complemented the development of Atlas, as it enabled a flexible workflow that could respond to the addition of new requirements. However, the author found it challenging to perform all of the roles required to maintain the Scrumban framework. As the author played the role of both the developer and the product owner, they were required to maintain the responsibilities of both roles. Therefore, in addition to maintaining the Scrumban board, creating the backlog, and handling the workflow, they also had to develop, test, and deploy the software. During busier periods, the Scrumban board was not updated to reflect current work, as the author prioritized development over management practices. In an attempt to maintain the Scrumban methodology, the author tried to update the Scrumban board before it began development. In future projects, the author will try to divide their time better to play different roles in an attempt to ensure the framework does not become too overwhelming.

X. Testing and Security

A. Testing Driven Development

For the development of Atlas, the author used test-driven development (TDD). TDD is a software development practice that requires failing unit tests to be created before the feature code is implemented (Beck, 2003) (Figure 21). Unit tests ensure that individual methods within the code display the intended behavior. In TDD, software requirements are first converted into test cases before the author codes the solution to the new functionality. This structure forced the developer to think about the intended behavior of the feature and the design of the application. In turn, this resulted in a simpler and more efficient code as the author prioritized the design of the external interfaces (Fowler, 2003). Furthermore, TDD improved the code's reliability and the author's understanding as it encouraged purposeful and consistent test cases (Beck, 2003). This mitigated the risk of poor code quality (R5). Future developers who work on the project could use these test cases as a starting point for understanding the intended behavior of the application.

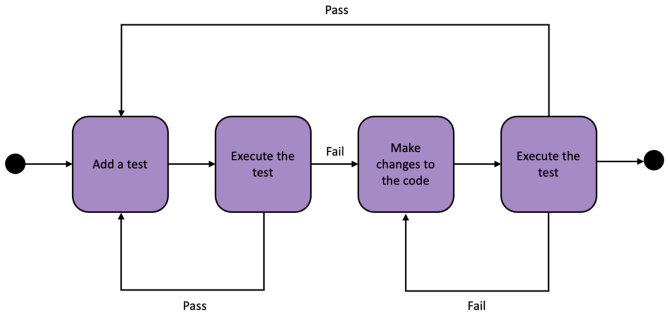

Fig. 21. The Test Driven Development cycle used in the development of Atlas enforced failing tests to be written before the code was implemented.

B. Functional Tests

In addition to unit tests, functional tests were implemented with every new feature. Functional tests validate the system works as intended by exercising the application as a whole (Juristo and Vegas, 2003). Thus, functional tests closely mimic the user journey and test the system's behavior at a high level. In order to run the functional tests, the application was containerized with docker. Docker is software that packages an application and its dependencies into a virtual container. Docker can be run on any operating system, as the virtual container ensures that tests are run consistently in an isolated environment. Therefore, using Docker reduces the chance of technology-specific bugs (Docker, 2021).

C. Continuous Integration Pipeline

It is important to regularly test an application during the development period to ensure no new bugs are introduced. The author introduced a continuous integration pipeline that automatically ran all unit and functional tests before a new feature was included in the master branch of the application. The author created the pipeline as a GitHub WorkFlow (GitHub, 2021). The WorkFlow ran all tests before a pull request could be merged into the master branch (Figure 22). The pipeline improved the code quality and reduced the number of manual steps required to release a new feature.

D. Non-Functional Testing

In addition to testing the functionality of the application, the author also tested its performance (NF3). It is important that the application could quickly respond to user commands to provide a smooth user experience. The author created a load test to simulate a user adding and removing 500 documents to and from the Atlas Web within a period of 15 seconds. The

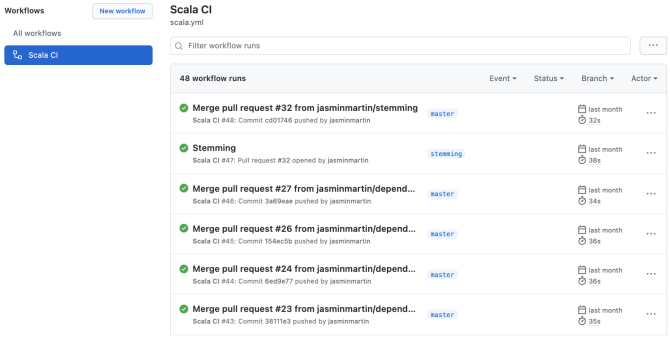

Fig. 22. The continuous integration pipeline for Atlas was created as a GitHub Workflow.

author used Gatling to create the load test. Gatling is an open-source load-testing tool that measures the performance of an application (Gatling, 2021). The resultant report revealed that Atlas was able to handle the load without throwing any errors or resulting in high latency above 0.1 seconds.

E. Docker Image Vulnerabilities

To improve the security of Atlas, the author scanned the application's docker image for vulnerabilities. The author used Docker Scan to discover that the application initially contained 38 vulnerabilities, 8 of which were rated highly severe. The vulnerabilities stemmed from Atlas's dependencies and transitive dependencies. These vulnerabilities could compromise the functionality or security of the application (Docker, 2021).

The docker scan revealed a highly severe vulnerability in the OpenJDK-11 dependency. This dependency contained a bug that enabled an unauthenticated attacker with network access via multiple protocols to compromise the Java SE (National Vulnerability Database – CVE -2021-2388, 2021). Another moderate-rated vulnerability in the GNU C dependency enabled a context-dependent attacker to cause a denial of service by exploiting regular expression bounded repetitions that bypassed the intended limits (National Vulnerability Database – CVE -2010-4051, 2021). The vulnerable packages were upgraded to the safer versions to remove any dependency-related security vulnerabilities (Figure 23).

```
➜  backend git:(new-notes) ✗ docker scan zettelkasten:0.1

Testing zettelkasten:0.1...

Organization:      undefined
Package manager:   rpm
Project name:      docker-image|zettelkasten
Docker image:      zettelkasten:0.1
Platform:          linux/amd64

✓ Tested 112 dependencies for known vulnerabilities, no vulnerable paths found.
```

Fig. 23. A docker scan demonstrating no vulnerabilities found in the dependencies or transitive dependencies of Atlas.

In addition to upgrading the vulnerable dependencies, the author also changed the application's base image to be distroless. Distroless images only contain the application code and the run-time dependencies. They lack the package manager, shells and operating system typical in a Linux distribution. Using a distroless image not only reduces the number of potential packages containing vulnerabilities but also makes the application less prone to surface attack, as there is no shell to access the container (Moore, 2017). By removing the shell, a potential attacker cannot read any secrets on the container or secure information persisted, such as user files (Figure 24). This also prevents the installation of any untoward packages. The distroless image also had the additional benefit of being 35% smaller.

```
jmn57@DSTML0501594 backend % docker ps
CONTAINER ID   IMAGE              COMMAND                  CREATED              STATUS              PORTS      NAMES
8950031db60e   zettelkasten:0.1   "/opt/docker/bin/zet…"   About a minute ago   Up About a minute   4024/tcp   pensive_agnesi
jmn57@DSTML0501594 backend % docker exec -it 8950031db60e bash
bash-4.4$ curl DELETE localhost:4024/file-body/cat
```

Fig. 24. An attacker could easily delete user documents in Atlas if it utilized a non-distress base image.

F. Dependabot Library Upgrades

To prevent further transitive dependency vulnerabilities, Dependabot alerts were enabled on the Atlas GitHub repository. The Dependapot actively scanned libraries for any vulnerabilities on a daily basis (Figure 25). The author could upgrade libraries from the GitHub interface, which would, in turn, run the continuous integration pipeline to ensure the upgrade did not inhibit the application functionality.

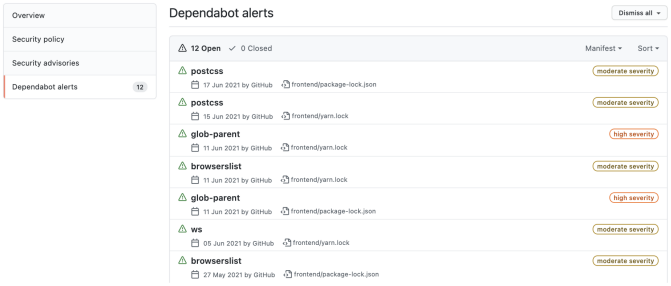

Fig. 25. Dependabot alerts signaling library upgrades are required in the Atlas repository to prevent vulnerabilities.

XI. Implementation

A. Local Artefact

The application code is open source; it is available to view, run, or extend by members of the public. The GitHub repository is complete with documentation in the readme.md, which advises users on how to run the application. The Latex project report and images are contained within the repository, along with instructions on how to generate this accompanying PDF.

B. Deployment Strategy

The author had planned to release Atlas onto the Google Cloud Platform (GCP) - their organization's chosen cloud computing provider. The cloud is a system of servers located in data centers all over the world (Pratim, 2018). The back-end and front-end of Atlas would be hosted in separate Kubernetes pods, which are scaled individually to meet client load. Client

requests would be directed with ingress that would enable access to the cluster.

Before the application was deployed to GCP, the author would modify the application architecture so that documents were stored in an Amazon S3 bucket (Figure 26). This would ensure documents could be accessed by multiple members of the organisation with consistent content.

To deploy new versions of Atlas to the cloud, the author would develop a continuous integration continuous deployment (CICD) pipeline. CICD pipelines automate the software delivery process by automatically building code, running tests, and deploying to different environments (Freeman, 2018). The former two processes were utilized in the development of Atlas with the aid of a GitHub WorkFlow. In order to include continuous deployment into the pipeline, a more appropriate CICD pipeline solution would be used such as Jenkins. The author's organization has configured Jenkins to deploy applications into a number of gated GCP environments. Each environment tests a different element of the service. For instance, a non-functional environment may test whether the application can handle simulated high loads, whilst an integration environment could enable developers to manually test features before they are released to production (Jenkins, 2016) (Figure 27).

C. Metrics and Monitoring

In order to provide a good level of service, it is important to maintain the performance and availability of a software application. Before Atlas is released into a production environment, the application must be able to generate and surface metrics. These metrics would be monitored to measure and alert the author to any problems with the user experience. Using the Kamon metrics instrumentation, the author could surface metrics that measured the traffic load, response time and response status codes (Kamon, 2021). These metrics could be visualized using a monitoring dashboard such as Grafana (Figure 28) (Grafana, 2021). Deviations of metrics above alerting thresholds could indicate problems with the infrastructure or application functionality. For example, a high percentage of requests resulting in 404 status code responses could indicate a bug within the application.

In addition to monitoring the API responses, it is also important to measure metrics related to computational resource

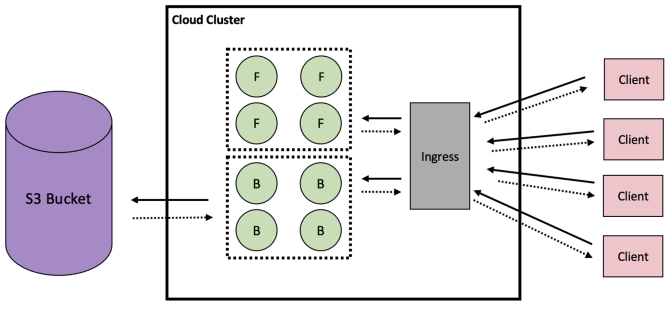

Fig. 26. Cloud-based architecture of Atlas. The green 'F' and 'B' circles represent the front-end and back-end pods.

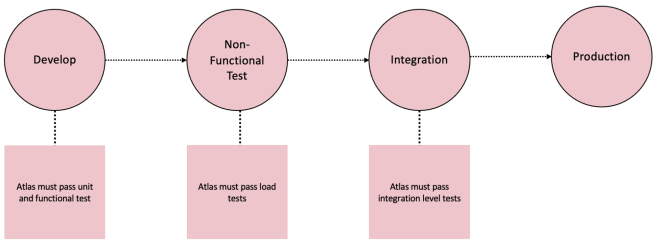

Fig. 27. Jenkins pipeline setup could release the Atlas application into different testing environments before it is released to production.

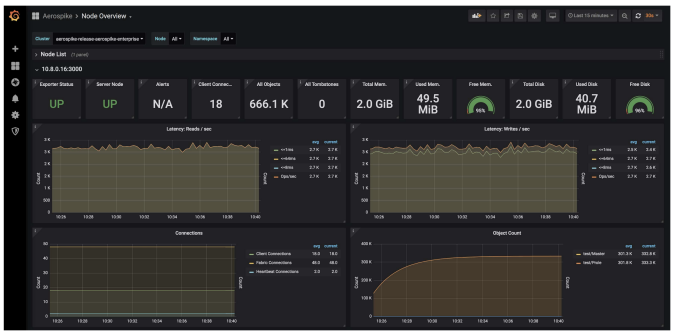

Fig. 28. An example of a Grafana monitoring dashboard that alerts the developer of problems in a production environment (Pujari, 2019).

usage. By understanding the CPU and memory usage of an application, the developer can predict the financial cost of running the application. Incorrect resource allocation could risk financial wastage or result in poor application performance (Pratim, 2018). Atlas would be deployed onto GCP, which provides an on-demand variability of computing resources. Although this means that pods can auto-scale to increase resource availability to meet load, it is still important to monitor resource usage as unexpectedly high resource usage may indicate issues with memory management in the application (Pujari, 2019).

D. Further Functional Requirements

As Atlas was developed using an agile management style, it would be possible to release the application at its current increment. However, future functional requirements could be added to improve the user experience (Table 10). The author did not fully develop the automatic keyword suggestion feature (F8). As this ability would make the application unique from competing knowledge management tools, prioritizing this feature could significantly increase product uptake and department interest. To implement this feature, the author would build upon their knowledge of the Play framework which would enable the application to efficiently store and recommend common keywords.

To use the aforementioned deployment strategy (Figure 26), Atlas would have to be modified so that it could maintain consistent content of shared documents (F11). This functionality would enable users in large organizations to easily update shared documents. As Atlas is currently configured to work

TABLE X
POTENTIAL FUTURE FUNCTIONAL REQUIREMENTS TO BE IMPLEMENTED INTO ATLAS.

No.	Functional Requirement
F8	To automatically suggest new connections
F11	To maintain consistent shared documents
F12	To collect feedback via an embedded survey

with local file systems, to achieve this, the application would have to be integrated with an Amazon S3 bucket, which would hold the state of the documents.

Although synthetic monitoring systems such as Dynatrace offer a thorough solution to monitoring user journeys, they require a subscription and maintenance cost (Dynatrace, 2019). Whilst the customer base of Atlas is low, the author could implement a cheaper solution: collecting feedback via embedded in-application surveys (F12). Such feedback could surface problems with the application's usability or uncover application bugs. Customer feedback widgets are one of the most common devices for gathering application feedback. Adding a short survey in the context of the application offers a convenient way for customers to submit feedback. Such methods are proven to have higher response rates than email surveys (Spiess et al., 2012).

XII. CONCLUSION

The development of Atlas provided the author with a mechanism for improving their project management and technical skills. Creating a greenfield application gave the author the opportunity to develop a product through the whole application life cycle, from soliciting requirements to designing and implementing the user interface and experience. The author was guided through the development process with their chosen Scrumban Agile methodology. This project management style provided a framework for the author to iterate upon the application whilst gathering and incorporating user feedback. Throughout the process, the author was able to interact and build working relationships with different members of the department whilst developing a tool that could improve productivity in the organization.

In total, the author achieved three of the four initial objectives, resulting in a functional iteration of Atlas. Although the application was not deployed into a production environment, the resulting local artifact forms a strong basis, which, with further work, could be used to improve the comprehension of the organization's documentation. User feedback on the current iteration suggests Atlas provided a clear user interface that offered a more intuitive experience than the hierarchical file system.

The author considers the project to be a success as it improved both their organizational and technical skills. In particular, they improved their front-end development abilities as they became more confident using the React framework. The author also learned more about natural language processing techniques and formed an interest in big data manipulation, which they would like to pursue further in the future.

XIII. FURTHER WORK

For Atlas to be used in an organizational setting, it must be deployed into a production environment. The aforementioned deployment strategy could be executed within an 8-week development period (Figure 29). First, a small functional change to the document storage would enable Atlas to be deployed onto the cloud (F11). The author would also implement metrics and monitoring to guarantee the application is running smoothly in production. As the author's organization already uses Jenkins pipeline infrastructure and GCP environments are already configured, the author estimates it would only take a further 4 weeks of development to deploy Atlas into production. Once deployment had been completed, the author would prioritize the automatic tag recommendation (F8) as the next developed feature as they have already researched big data processing frameworks.

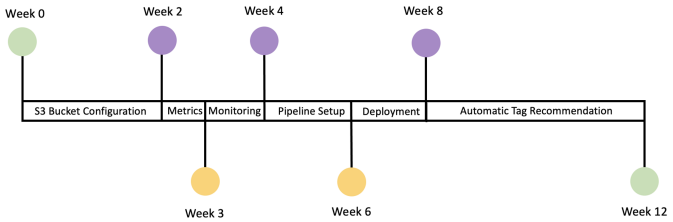

Fig. 29. The author would dedicate the next 8 weeks of development to ensuring Atlas was ready to be deployed into production. A further estimated 4 weeks could then be used to develop the automatic tag recommendation feature.

In addition to developing the application functionality and infrastructure further, the author would also dedicate effort to introducing more developers in the organization to Atlas. As the product will remain with the organization, it is important to future-proof the application by ensuring other developers have the context of the code and understand the related documentation. The author could engage more developers in the project by demoing the deployed product in the department showcase and encouraging audience members to take part in the next stages of development.

XIV. REFLECTIVE STATEMENT

Over the course of the synoptic project, the author built upon their behavioral and technical skills. Both the project management and technical development of Atlas gave the author the opportunity to exercise and evaluate the abilities defined in the apprenticeship standard. Drawing upon examples from the development of Atlas, here, the author reflects on how three specialist technical competencies and three core behavioral skills were improved over the duration of the course (Table 11). Furthermore, the author applies the Gibbs behavioral model to their personal experiences in an attempt to understand how they can further their abilities (Gibbs, 1997). Evidence of other apprenticeship standard skills can be found in Appendix 3.

TABLE XI
THE SPECIALIST TECHNICAL SKILLS AND CORSE BEHAVIOURAL SKILLS THAT WERE IMPROVED BY THE AUTHOR OVER THE DURATION OF THE SYNOPTIC PROJECT.

Id	Skill Description
STS-1	Create effective and secure software solutions using contemporary software development languages to deliver the full range of functional and non-functional requirements using relevant development methodologies.
STS-2	Test code to ensure that the functional and non-functional requirements have been met.
STS-3	Deliver software solutions using industry-standard build processes and tools for configuration management, version control, and software build, release, and deployment into enterprise environments.
CBS-1	Makes concise, engaging, and well-structured verbal presentations, arguments, and explanations.
CBS-2	Applies analytical and critical thinking skills to Technology Solutions development and to systematically analyze and apply structured problem-solving techniques to complex systems and situations.
CBS-3	Able to conduct effective research, using literature and other media, into IT and business-related topics.

A. STS-1: Developing Functional and Non-Functional Requirements

The author selected and applied the Scrumban agile framework to ensure the functional and non-functional requirements were iteratively developed using Scala in the synoptic project. They felt Scrumban was appropriate because it complemented both the time period and the size of the project. However, the author failed to consider the fact that they were the sole people in the framework and, as a result, spent a lot of time playing the roles of the scrum master, product manager, and developer. Consequently, they felt overwhelmed at times with the amount of organization and administration needed to maintain all three roles.

This experience taught the author that process-heavy frameworks may not be appropriate for individual developers and may require adaptations. Despite this, they do not regret applying an organizational framework, as it gave them a taste of how the product management style could operate in the workplace. The author felt that if they were to solely develop a product again, they would take inspiration from a product management style rather than implement the whole framework. For instance, if they chose Scrumban again, they could utilize the Kanban board and progressive backlog but not worry about estimating tickets. Moreover, if a project management style did not fit the project, the author would not be afraid to step outside of the framework before it became overwhelming.

B. STS-2: Testing Functional and Non-Functional Requirements

As part of their synoptic project, the author engaged in test-driven development (TDD) to test the functional requirements via unit tests. The author felt that TDD enabled them to build confidence in their code as it ensured each method was tested. Furthermore, they were surprised to find that the style also encouraged them to plan and organize features better. The author enjoyed TDD as it acted as a framework for reviewing and improving their own code. The author has since brought this practice into their workplace and has found it has helped to align coding styles when pairing.

To ensure non-functional requirements were met, the author introduced load-style Gatling tests. Although the load test ensured the application could cope with spikes in traffic, on reflection, the author would spend less time implementing such a large-scale framework in the future. As the application was only used simultaneously by a few people, the author did not believe such thorough testing was needed. Instead, if the author were to test the non-functional requirements again, they would take the opportunity to measure the security and user experience of the application. The author realized that they had not collected enough metrics on these areas and would be interested in researching industry practices to collect these metrics. Moreover, as the author is predominantly a back-end developer, they would be curious to see how front-end elements such as user journey experiences could be improved.

C. STS-3: Delivering Software Solutions

The author particularly enjoyed delivering software solutions by engaging in all areas of the product life-cycle, from build to deployment. As the author had predominately been involved in the build side of the life-cycle and already knew how to use version control and configuration management in the workplace, they felt the apprenticeship helped them understand the whole life-cycle process and appreciate how the solution is delivered to customers. In turn, the author felt more engaged with the users of their product as they understood how they interacted with the application from a technical perspective. However, at times, they did feel constrained by the organisation's choice of technology. For instance, the organization favored the Jenkins pipeline instead of CircleCI – which is a more common industry solution (Frankinson, 2020). Instead of becoming frustrated, the author learned to accept the limitations of the organization's choices and, instead, tried to understand why certain technologies had been selected from a business approach. This helped them understand the build processes within the business.

D. CBS-1: Verbal Communication

In order to gather feedback to incorporate into the design of Atlas, the author showcased the application in a series of presentations. In order to ensure the presentations encouraged detailed feedback, the author researched how to tailor the content to the demographic of the audience. For instance, in presentations with a non-technical audience, the author ensured they used non-technical language and focused on high-level concepts rather than technical implementations. The author also improved general presentation skills by reducing the amount of text in the presentation to make it more concise and by incorporating more diagrams to maintain the interest of the audience.

The author was happy with their progress in improving their verbal communication skills, as they felt they were able to better showcase their product, which in turn produced higher quality feedback. The author had not previously conducted focus groups or interviews, so the synoptic project gave them the opportunity to structure explanations in a less rehearsed format. A key difference the author found between these formats was that the author's role was more focused on persuading and engaging the audience through the process rather than just explaining. The author was able to use their leadership skills to guide the discussions of others rather than

detail arguments upfront. Moreover, the showcases improved the author's confidence, and the positive feedback from the audience motivated them. If the author were to conduct product research again, they would be more likely to showcase the product to larger audiences.

E. CBS-2: Applying Critical Problem Solving to Technology Solutions

The author was tasked with architecting the project design. As this was a greenfield project, they were able to choose the technology and infrastructure that would support the application in a production environment. The author researched different languages and frameworks that could support the application. They ensured the technologies were able to efficiently and quickly generate the graph as well as display it in an appropriate fashion. The author chose React Typescript as a front-end framework after comparing it to other popular frameworks. The author enjoyed using React, as they found it intuitive to use and were supported by a large amount of online documentation. Moreover, React allowed the author to create reusable components, which reduced the amount of code needed to render the graphs. The author learned not to become overwhelmed with the number of technical solutions available and, instead, research and trial the technology to see if it was appropriate.

The author was also able to improve their back-end abilities by researching natural language processes which would enable them to efficiently generate the Atlas graph. The author used a tokenizing algorithm that simplified the ability to select common keywords in the application. The author took an interest in natural language processing and also used the Porter stemmer algorithm to standardize the document file names. The author learned that by adapting well-known algorithms, they were able to build upon existing computer science solutions to complex problems.

F. CBS-3: Conducting Research

In the early stages of development of the synoptic project, the author researched a series of literature to form an understanding of how to efficiently create their application. The author conducted a literature review on related content, such as project management techniques and technical processes. From their previous experience, the author knew to use tools such as Google Scholar to find relevant papers. However, the author found it difficult to find relevant technical content and, instead, found looking at online articles as a starting point to give a general overview of a technical matter to be more efficient. The author was initially surprised this made the research more effective but found it significantly decreased the level of understanding required to approach a technical subject.

In order to conduct research into the business needs, the author analyzed the current procurement processes. The author was required to design and execute a primary research plan within their organization. This exercised the author's planning and investigation skills, as they were tasked with finding the appropriate teams to communicate with and organize meetings. On reflection, the author would begin the research process earlier into the project timeline as they found it time-consuming to schedule talks with different organization members. Overall, the experience taught them to use several different information sources to build a more rounded opinion.

XV. Glossary

Agile: A software development methodology focused on iterative cycles of development. Requirements and results are continuously evaluated to quickly respond to change.
API: Application Programming Interface is a software intermediary that enables two applications to communicate with each other.
Atlas Web: The main user interface of the Atlas application that displays the file system as a graph.
Auto-Scaling: a cloud computing process that dynamically adjusts the amount of computational resources with traffic load.
CICD: Continuous integration continuous deployment is the combined practice of building and releasing code into a production environment.
Cloud Computing: An on-demand availability of computer system resources and data storage that does not require direct active management.
Cluster: A shared group of networking resources for Kubernetes pods.
CORS: Cross-Origin Resource Sharing is an HTTP-header-based mechanism that secures browsers by limiting the domains and ports from which resources can be received.
CPU: The Central Processing Unit is the electronic circuitry that runs computer systems.
CRUD Request: Create, read, update, and delete requests are the four functions necessary to implement mutable persistent data.
cURL Request: client URL is a command line tool that can be used to make requests to a website.
Docker: a containerized platform that combines the application source code, libraries, and operating system into a specific environment.
Extreme Programming: A software development methodology focused on improving software quality and the speed of response to changing customer requirements.
GCP: Google Cloud Platform is a provider of cloud computing resources for developing, deploying, and operating software applications.
Github: A website that hosts software and acts as a version control system.
Github Workflow: a configurable automated process that is run on GitHub before code is merged into the master branch.
Ingress: an API object that provides external access to a Kubernetes cluster.
Jenkins: An automation pipeline that enables software to be built, tested, and deployed into different environments.
Kanban: An agile workflow management method for defining, managing, and improving software development.

Kubernetes: A container-orchestration system for automatically deploying, scaling, and managing applications in pods.
Lemmatisation: The process of grouping together the inflected form of a word. The lemma of running, runner, and runs is run.
Pod: Point of delivery is a module of application components, storage, and network resources that work together to deliver a service.
Porter Algorithm: A stemming algorithm used to standardize the suffixes of words.
Roam: An online knowledge management tool.
S3 Bucket: A cloud storage system that utilizes Amazon Web Services (AWS). Also known as the Simple Storage Service.
Scrum: An agile framework for developing and delivering products by generating adaptive solutions that help organize teams and organizations.
Scrumban: An agile development methodology that is a hybrid of Scrum and Kanban.
Spike: An extreme programming practice whereby a developer allocates a time period to research or prototype a technical solution.
Test Driven Development: a style of programming whereby failing tests are written first to direct the implementation.
Transitive Dependency: A software dependency that is introduced by the libraries that the program references.
Zettelkasten Algorithm: A method for organizing information into tree-based structures.

REFERENCES

[1] A. Abraham, "Cheatsheet Differences Scrum vs Scrumban vs Kanban," Medium, 2021. [Online]. Available: https://medium.com/agileinsider/comparison-of-scrum-vs-scrumban-vs-kanban-1d1d2b9a9fd5. [Accessed: Oct. 3, 2021].

[2] A. Afzal, J. Wasif, R. Torkar, and R. Feldt, "A systematic review of search-based testing for non-functional system properties," *Inf. Softw. Technol.*, vol. 51, no. 6, pp. 957–976, 2009.

[3] Agile Dictionary, "Spike — The Agile Dictionary," 2021. [Online]. Available: http://agiledictionary.com/209/spike/. [Accessed: Oct. 3, 2021].

[4] S. Augustine, B. Payne, F. Sencindiver, and S. Woodcock, "Agile project management: Steering from the edges," *ACM*, vol. 48, pp. 85–89, 2005.

[5] K. Beck, *Test-driven development: by example*. Addison-Wesley Professional, 2003.

[6] G. Chowdhury, "Natural language processing," *ARIST*, vol. 37, pp. 51–89, 2005.

[7] K. Clark, "A $200 Million Seed Valuation for Roam Shows Investor Frenzy for Note-Taking Apps," *The Information*, 2021. [Online]. Available: https://www.theinformation.com/articles/a-200-million-seed-valuation-for-roam-shows-investor-frenzy-for-note-taking-apps?shared=931cbf4ce58ed9bd. [Accessed: Oct. 3, 2021].

[8] J. Coleman, E. McTigue, and J. Dantzler, "What makes a diagram easy or hard? The impact of diagram design on fourth-grade students' comprehension of science texts," *The Elementary School Journal*, vol. 119, no. 1, pp. 122–151, 2018.

[9] S. Cottrell, *Dissertations and Project Reports*. Basingstoke, UK: Palgrave Macmillan, 2014.

[10] F. Damerau and N. Indurkhya, *Handbook of Natural Language Processing*, 2nd ed. Boca Raton, FL, USA: Taylor Francis, 2010, pp. 72–78.

[11] C. Dawson, *Projects in Computing and Information Systems*. Harlow, UK: Pearson Education, 2015.

[12] N. Doherty, D. Champion, and L. Wang, "An holistic approach to understanding the changing nature of organisational structure," *Inf. Technol. People*, vol. 23, no. 2, pp. 116–135, 2010.

[13] T. Dybå, T. Dingsøyr, and N. B. Moe, "Software Project Management in a Changing World," vol. 2, no. 2, pp. 430–439, 2014.

[14] N. Eliason, "Roam: Why I Love it and How I use it," Nateliason, 2020. [Online]. Available: https://www.nateliason.com/blog/roam. [Accessed: Oct. 1, 2021].

[15] K. Finley, "Microsoft Previews New JavaScript-Like Programming Language TypeScript," *Tech Crunch*, 2012. [Online]. Available: https://medium.com/agileinsider/comparison-of-scrum-vs-scrumban-vs-kanban-1d1d2b9a9fd5. [Accessed: Oct. 7, 2021].

[16] T. Forte, "How To Take Smart Notes: 10 Principles to Revolutionize Your Note-Taking and Writing - Forte Labs," Forte Labs, 2021. [Online]. Available: https://fortelabs.co/blog/how-to-take-smart-notes/?fbclid=IwAR0Dbtdio3oYPicwAaFToU1EiEBrXGiQavUxv8lR2CSMaupAKQASeT-Ckts. [Accessed: May 14, 2021].

[17] M. Fowler, *UML Distilled: A Brief Guide to the Standard Object Modeling*, 2nd ed. Addison Wesley, 2003, pp. 32–45.

[18] M. Fowler and J. Highsmith, "The Agile Manifesto," 2000.

[19] S. Ghemawat, H. Gobioff, and S. Leung, "The Google file system," *ACM SIGOPS Operating Systems Review*, vol. 37, no. 5, pp. 29–43, 2003.

[20] A. Gibbs, "Focus groups," *Social Research Update*, vol. 19, no. 8, pp. 1–8, 1997.

[21] B. Gibson, "Building a Slip-Box," 2021. [Online]. Available: https://www.barryjohngibson.com/post/down-the-rabbit-hole-building-a-slip-box-or-zettelkasten. [Accessed: May 14, 2021].

[22] GitHub, "Dagrejs/dagre," 2021. [Online]. Available: https://github.com/dagrejs/. [Accessed: Oct. 3, 2021].

[23] Grafana.com, 2021. [Online]. Available: https://grafana.com/. [Accessed: Oct. 3, 2021].

[24] G. L. and T. Raghu, "Determinants of Mobile Apps' Success: Evidence from the App Store Market," *J. Manag. Inf. Syst.*, vol. 31, no. 2, pp. 133–170, 2014.

[25] M. Hegarty, P. A. Carpenter, and M. A. Just, "Diagrams in the comprehension of scientific texts," *Handbook of Reading Research*, vol. 2, pp. 641–668, 1991.

[26] R. Hoda, J. Noble, and S. Marshall, "Agile project management," in *New Zealand Computer Science Research Student Conference*, NZCSRC, 2008.

[27] R. Jeffries, "RonJeffries.com," 2021. [Online]. Available: https://ronjeffries.com/. [Accessed: Oct. 3, 2021].