

Increasing User Engagement on the Archive of Formal Proofs

Carlin MacKenzie



Minf Project (Part 2) Report
Master of Informatics
School of Informatics
University of Edinburgh

2022

Abstract

Isabelle is a language for formalising mathematical proofs in code and the Archive of Formal Proofs (AFP) collects mechanically checked proofs in a similar way to an academic journal. In Part One we redesigned the AFP and re-implemented it with the static site generator Hugo. This effort was successful and is in the process of being officially integrated.

In this part we aim to increase user engagement in the AFP with the introduction of social features. We add comments, notifications, user accounts and profiles. We also allow users to customise how they use the site with the ability to pin topics and web feeds. This is possible with the addition of a dynamic server which provides an API and a database. So that we can preserve user privacy we do not rely on third-party services. Our resulting implementation was evaluated with users who found it to be an improvement. Finally, we improved the design based upon suggestions from the user evaluation.

The result of this project is a more social and engaging Archive of Formal Proofs which meets the constraints as set out.

Research Ethics Approval

This project was planned in accordance with the Informatics Research Ethics policy. It did not involve any aspects that required approval from the Informatics Research Ethics committee.

The participants' information sheet and a consent form are included in the Appendix.

Declaration

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

(Carlin MacKenzie)

Acknowledgements

I wish to thank my supervisor Jacques Fleuriot and my co-supervisor James Vaughan for their constant support with this project. It was a pleasure to work with the both of them. Additionally I would like to thank my mum and dad for their support which has allowed me to be where I am today.

Finally I would like to thank PG Tips Decaf, Sainsbury's Extra Strong Red Label, Tick Tock Earl Grey Rooibos and Sea Dyke Brand Jasmine tea for their support. I wish I had counted the hundreds of cups of tea that were consumed, which without doubt were vital to the completion of this project.

Table of Contents

1	Introduction	1
1.1	Previous Work	1
1.1.1	Integration into the Official AFP	1
1.2	Part Two	2
2	Background	4
2.1	Online Communities	4
2.1.1	Mizar	4
2.1.2	Lean	4
2.1.3	Coq	5
2.1.4	arXiv	5
2.1.5	Wikipedia	6
2.2	Archive of Formal Proofs	6
2.2.1	Community	6
2.2.2	Development Philosophy	7
2.3	User Engagement	8
3	Design	10
3.1	Feature Set	10
3.2	Prototyping	11
3.2.1	Comments	11
3.2.2	Notifications	12
3.3	Design Considerations	13
3.3.1	Feedback Messages	13
3.3.2	Icons	13
3.3.3	Responsive Design	14
3.3.4	Web Feeds	14
3.4	Conclusion	14
4	Implementation	16
4.1	Server Architecture	16
4.1.1	Approach	16
4.1.2	Structure	17
4.1.3	Hosting	18
4.2	Database	18
4.2.1	Access	19

4.2.2	Schema	20
4.3	API	20
4.3.1	Routing	21
4.4	User Accounts	21
4.4.1	Authentication	22
4.4.2	Profile	22
4.4.3	User URL	23
4.5	Statistics	23
4.6	Commenting on Entries	24
4.7	Notifications	25
4.7.1	Initial Implementation	25
4.7.2	Comment Replies	26
4.7.3	Unread Counter	26
4.8	Customising Topics	27
4.9	Feedback Messages	28
4.10	Web Feeds	28
4.11	Error Handling	28
4.11.1	Redundancy	28
4.12	Privacy	29
4.12.1	Cookies	29
4.13	Conclusion	29
5	Evaluation	30
5.1	User Evaluation	30
5.1.1	Design	30
5.1.2	Results	31
5.1.3	Conclusion	33
5.1.4	Resulting Changes	33
5.2	Technical Evaluation	33
5.3	Performance	34
5.4	Maintenance	35
5.5	Conclusion	35
6	Conclusion	36
6.1	Suitability for Production	36
6.2	Future Work	37
6.3	Concluding Remarks	37
	Bibliography	38
	A Paper Prototypes	41
	B Commands to Host the Extended AFP	46
	C Screenshots of the Extended AFP	47
	D Evaluation	52

D.1	Participants' Information Sheet	52
D.2	Participants' Consent Form	56
D.3	Script for the Evaluation	58
E	Poster	60

Chapter 1

Introduction

This report comprises the second part of a project to improve the Archive of Formal Proofs (AFP). The AFP is the central repository for Isabelle mathematical proofs and has not been visually or functionally updated since its release in 2004. We summarise Part One before introducing our work in this report.

1.1 Previous Work

Part One [23] principally involved the recreation of the Archive of Formal Proofs website using the static site generator Hugo and the subsequent redesign (Figure 1.2). The resulting website was faster to generate, had a smaller file size and was more functional for the user as we made improvements to search, navigation and code browsing. Both the current [24] and redesigned AFP were validated by use of a survey, and the latter was additionally evaluated using a walk-through with the users.

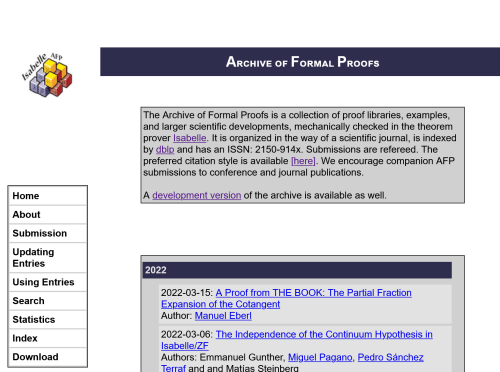


Figure 1.1: Current Archive of Formal Proofs



Figure 1.2: Part one redesign

1.1.1 Integration into the Official AFP

After submission of the report, the redesign was announced on the Isabelle mailing list on May 3rd¹. We were already in contact with a maintainer of the Archive of Formal

¹<https://lists.cam.ac.uk/sympa/arc/cl-isabelle-users/2021-05/msg00011.html>

Proofs at the University of Munich and he informed us that they wanted to move to a new static site generator and that they liked the work that we had done so far. We agreed that they could use the design and that we would help with integration.

The integration has led to several changes, some implemented by us and others by them. For example, they rewrote the Python scripts in Scala and integrated them into the Isabelle build system. They also re-implemented the theorem viewer so that it loaded the theories dynamically, lowering the load on the client and server. We provided help with accessing data from Hugo and design improvements for the theory browsing page and mobile design. As the contributions must be distinct between the two parts of this project, we do not focus on the work done for this integration in the current report.

1.2 Part Two

The results described in the previous section provide a solid foundation for future improvements.

Motivation While the AFP has been redesigned and re-implemented, it is functionally the same as it was previously. Customisation of the site is not possible, and the community is organised externally. Users email authors directly with their questions, making the information inaccessible.

Objective We aim to make the site more engaging for users and encourage community growth by extending the AFP with dynamic features such as comments and user profiles. The extension should not rely on third-party services and should be congruous with the static site nature of Part One.

Contribution We present several contributions:

- *Dynamic back-end:* We preserve the static Apache server [11] and add a Node.js server [8] to provide an API. We use a SQLite database [34] to store state.
- *User Profiles:* We add accounts to the AFP, which allow people to make comments, customise the AFP and express themselves on their profile. We authenticate users using Passport.js [17].
- *Comments:* The comment server Isso [41] was used to provide comments on entries and we integrated this with our account system.
- *Notifications:* We introduce notifications to enable users to see replies to their comments. This is built on top of Isso, and we add a live-updating unread counter so that they are notified promptly.
- *Customisation:* The ability to customise the order of topics was added to allow users to fit the site to their needs.
- *Evaluation:* The extension of the AFP was evaluated by the same study group from Part One to understand if the project was successful.

Organisation We start by describing the online communities of theorem provers on the Internet in Chapter 2. Next we describe the design process and the features that we will focus on in Chapter 3. We discuss the implementation in Chapter 4 before evaluating it in Chapter 5. We conclude in Chapter 6.

Chapter 2

Background

This chapter provides the context for the work that follows by describing other academic online communities before detailing the communities around the AFP. We also perform a literature review of the research surrounding user engagement.

2.1 Online Communities

The history and mathematical libraries of Mizar, Coq, Lean and Isabelle were described in Part One. Here we discuss the community spaces that exist for each proof assistant.

2.1.1 Mizar

Mizar was one of the first proof assistants and relies on a mailing list for community discussion. This mailing list is called the “Mizar Forum” and in recent years is mainly used for advertisement (“Call for Papers” and new tools). There also exists an email address which aims to respond to help queries within 48 hours.

There does exist the “Association of Mizar Users” (SUM, in Polish), however it has little online presence. It seems that this community exists mainly offline at its host institution, the University of Bialistok, and one can only join the society by being introduced by two current members¹.

2.1.2 Lean

Lean uses both GitHub [15] and Zulip [22] to gather its online community. There is one GitHub repository for the entire Lean proof library². GitHub Issues are used to suggest features and improvements and these are subsequently categorised with labels and triaged with GitHub Projects. Pull requests are used to contribute to the project, and there are 383 open to 12,402 closed pull requests.

¹<http://www.mizar.org/sum/>

²<https://github.com/leanprover-community/mathlib>

Zulip [22] is a synchronous chat client similar to Slack [33]. It is organised into *streams*, such as “#new members” and “#Is there code for X?” and *topics*, such as “#new members > zero not 0?” and “#Is there code for X? > ring.inverse is continuous”. Users can subscribe to streams they are interested in and mute topics that they are not interested in within streams. Topics allow for conversations to be contained and asynchronous. It is free and open source and can be self-hosted or hosted on Zulip’s servers. Zulip additionally suits maths communities by allowing for MathJax and Markdown syntax.

Lean’s use of Zulip has been highlighted in a case study by Zulip [42] and mentioned in Wired’s profile of the community [18] (originally published in Quanta Magazine [19]). The case study describes how Zulip allowed for ten members to co-author a paper remotely, verifying a development in condensed mathematics by the Fields medallist Peter Scholze. There are currently around 450 users checking Zulip at least every 15 days and there are roughly 4,000 messages sent per week.

2.1.3 Coq

Coq uses many platforms for its online community [37]. First, each package in its library is a GitHub repository and so each has space for discussion and issues. This means that questions and errors can be localised to the package itself.

For more general queries, there is a forum for Coq hosted by Discourse. There are boards for help, development, Coq itself, as well as boards for specific languages like French and Chinese.

Previously, Coq relied on IRC for synchronous chat, but has recently moved to Zulip. There are around 180 users who check Zulip at least once every 15 days and around 1,000 messages are sent per week.

2.1.4 arXiv

arXiv [40] is an open-access e-print archive for scientific papers. While it does allow for users to create accounts to manage their submissions and authorship, there are limited social mechanics on the site. For example, authors do not have profiles and there are no discussion sections.

ArXiv does provide some ways to customise the user’s experience, such as web feeds and “catchup”. Web feeds allow users to subscribe to a feed of new preprints in an external reader. Users can see all new papers since a certain date with the catchup feature.

Through the arXivLabs project, external organisations can contribute interactive widgets to be on arXiv pages. This contributes to user engagement with the site as users can choose to explore these added features such as exploring citations or related papers.

2.1.5 Wikipedia

Wikipedia [12] is the largest and most popular reference work on the Internet [1] and has many social features. Of most importance, every page has an associated talk page. This serves as a space to discuss the quality of the entry and ask questions of the people who are editing it. Users also have pages, which can be thought of as their profile. These pages also have associated talk pages, and serve as a space to leave messages for other members. Users also associate themselves into editing groups with WikiProjects. These serve to organise effort and highlight areas that need more attention.

2.2 Archive of Formal Proofs

The structure and features of the Archive of Formal Proofs were described in detail in Part One. As there are currently no social elements on the AFP, we describe the social spaces that exist externally.

2.2.1 Community

Since the release of the AFP in 2004, 424 authors have contributed to the AFP with an average of thirty-five new authors per year (2017-2021). It would be nice to estimate the size of the community by the number of downloads of each Isabelle release or the number of subscribers to the mailing list, however these statistics are not available.

2022/03 31 mails

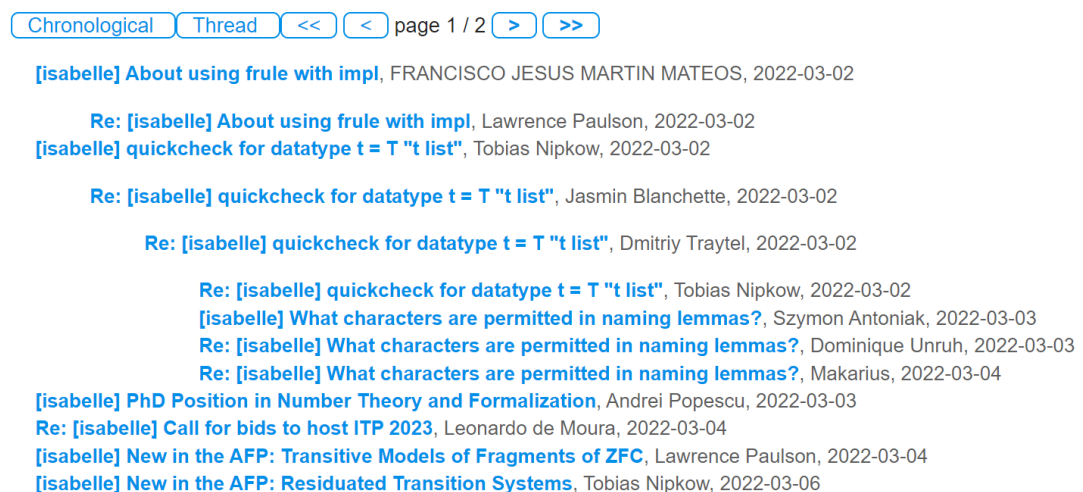


Figure 2.1: A selection of emails from the Isabelle-users mailing list from March 2022

The AFP primarily communicates via two mailing lists. The primary list is Isabelle-users³ which serves as a place for asynchronous discussion about problems, solutions, and results. Also new entries in the AFP are advertised here as well as related conferences and job postings (see Figure 2.1). The number of messages to the mailing list varies throughout the year, averaging between 40-200 emails per month. The second

³<https://lists.cam.ac.uk/pipermail/cl-isabelle-users/index.html>

mailing list is `isabelle-dev`⁴ which concerns itself with the development of Isabelle and the AFP. Fewer emails are sent here, averaging between 1-50 per month.

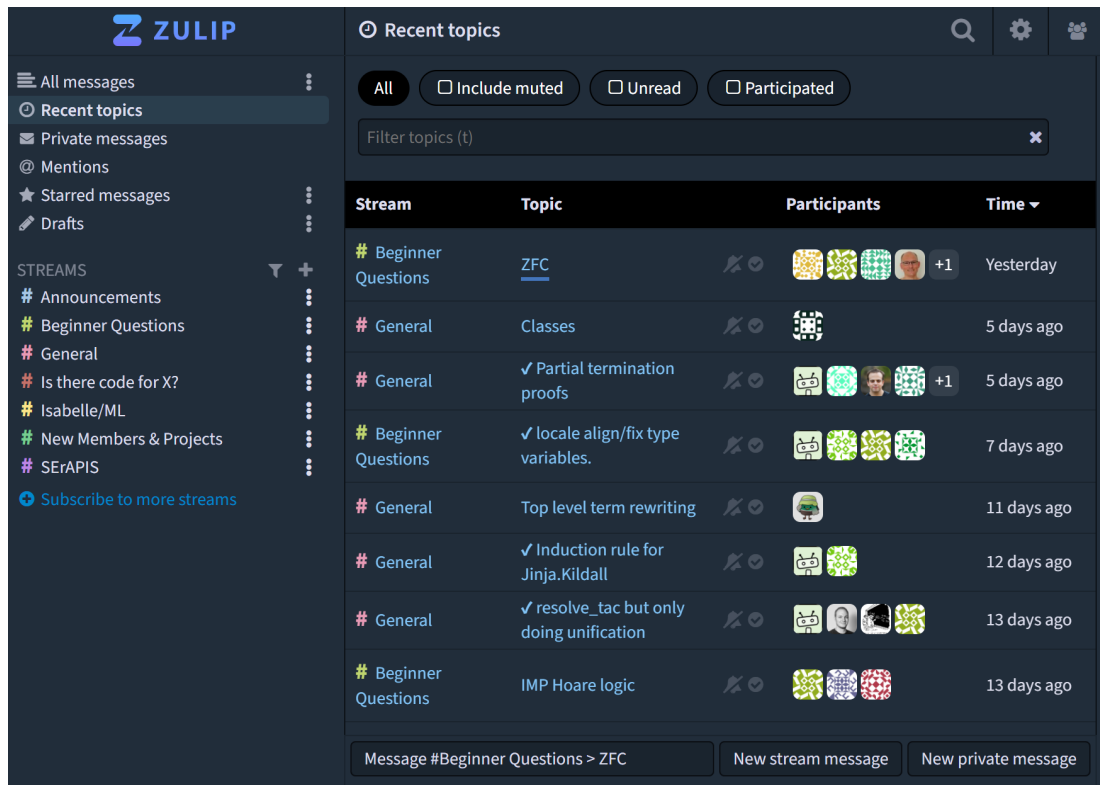


Figure 2.2: The Isabelle Zulip showing the topics view

There also exists some communities outside the mailing lists, which are linked from the Isabelle website. These are the Zulip and the StackExchange. As mentioned in the Section 2.1.2, Zulip is a real-time chat client similar to IRC and Slack. It can be preferable to discussion on a mailing list as it is immediate, and lively threads can be muted by those not interested. There are 422 members of Zulip with around 70 active users, defined as being online at least once every 15 days. Around 200 messages are sent per week.

On the other hand, StackExchange [21] is a question and answer forum which has a child website for proof assistants⁵. Each of the provers mentioned have a tag on the site and there are currently 981 questions under the “isabelle” tag. Many users can contribute to a solution to a problem and discuss it in the comments.

2.2.2 Development Philosophy

Isabelle and the AFP both have both been intentionally designed with a DIY mentality — almost everything has been created from scratch. This means that no third-party services are depended upon and very few libraries are used. This has impacted how we approach this project, as we do not want to store data with third parties or introduce

⁴<https://mailmanbroy.informatik.tu-muenchen.de/pipermail/isabelle-dev/>

⁵<https://proofassistants.stackexchange.com>

The screenshot shows the 'Proof Assistants Beta' page on Stack Exchange. The main heading is 'Explore our Questions' with an 'Ask Question' button. A navigation sidebar on the left includes 'Home', 'PUBLIC', 'Questions', 'Tags', 'Users', and 'Unanswered'. The main content area displays three questions with their respective statistics and tags.

Question	Votes	Answers	Views	Tags	Author	Modified
How does a logical framework become a meta-logical framework?	7	1	91	isabelle, logical-frameworks, isabelle-hol, meta-logical-framework	Guy Coder	2,223 modified Feb 25 at 17:34
How to prove Gauss summation of integers with proof assistants using forward reasoning?	8	2	107	isabelle, forward-proof, backward-proof	tinlyx	1,196 modified Feb 22 at 18:58
Standard ML is used for many proof systems. Is there a recommended implementation to use for Isabelle?	4	2	86	isabelle, standard-ml, build	Lawrence Paulson	473 answered Feb 21 at 16:09

Figure 2.3: The theorem proving Stack Exchange where Isabelle questions are posted

unknown points of failure. On top of this, common software engineering practices are subverted so that they can be done in an “Isabelle” way. For example, the project does not use the now almost universal version control software git [30] but instead uses Mercurial [6].

In Part One we established certain practices that we aim to maintain in this part. First, we chose Hugo as our site generator as it has a simple approach to content where data is stored in text files. This makes data management easier as it is not tied into a proprietary format. Second ...

2.3 User Engagement

In 2016, Garrett et al. performed a literature review of web design and user engagement by analysing the top one hundred papers under “(design) AND (usability) AND (websites)” [13]. They found twenty distinct attributes which influence user engagement and seven of them (1, 2, 3, 4, 5, 8, 17) were in at least 30% of their studies (their arbitrary cut-off to select for a set of guidelines). Of these twenty, we addressed thirteen of these in Part One and four were already present in the original AFP.

1. organization.	6. memorable elements	11. consistency	16. strong user control capabilities
2. content utility.	7. valid links	12. <i>accuracy</i>	17. readability.
3. navigation.	8. simplicity.	13. loading speed	18. efficiency
4. graphical representation.	9. <i>impartiality</i>	14. security /privacy	19. scannability
5. <i>purpose.</i>	10. <i>credibility</i>	15. interactive	20. learnability

Table 2.1: Design elements identified by Garrett et al. *Italics* indicate elements which were already provided by the current AFP. **Bold** indicates elements which were not addressed in Part One

Chapter 3

Design

From our work in Part One, we have a strong design aesthetic in place and as such our design in this part must not degrade it. We do not want to add clutter or inconsistent design elements that make it obvious which parts have been “bolted-on” so to say. As Nielsen identifies, more features bring more complexity and features should be carefully chosen so as to not overwhelm the user [27]. Therefore our goal in our design is that “everything should be made as simple as possible, but not simpler”, a quotation often attributed to Einstein.

3.1 Feature Set

As we extended the AFP, we needed to have an idea of the features we were accounting for, before designing how they would interact and function. In Section 2.3, we discussed Garrett et al.’s literature review which identified twenty features which contribute to user engagement. Of these twenty, three have not yet been accounted for. These are defined as such:

14. security/privacy - does the website securely transmit, store, and display personal information/data,
15. interactive - can the user interact with the website (e.g., post comments or receive recommendations for similar purchases)
16. strong user control capabilities - does the website allow individuals to customize their experiences (such as the order of information they access and speed at which they browse the website),

We keep our system secure by adding authentication (Section ??) and user data private by not relying on third-party services. In terms of interactive features, we chose to add comments (Section ??) and user profiles (Section ??). Finally, we allow users to control their experience by customising the topics page (Section ??) and by adding web feed support (Section ??).

While features such as email validation and management, admin features (including moderation) and submission would bring great benefit to the project, we decided

that they were out of scope as they would not contribute directly to increasing user engagement. Additionally, in Part One of the project we imagined creating a nicer interface to edit entries after they are submitted. However this goal is misguided as the AFP is an archive and entries are intentionally static and should not be edited after release.

3.2 Prototyping

In Part One we almost exclusively relied on paper prototyping. In this project we have to not only consider how each page will look, but also the user's journey when signing up and editing their profile. As such we used a combination of paper and interactive prototypes using Figma (see Figure 3.1).

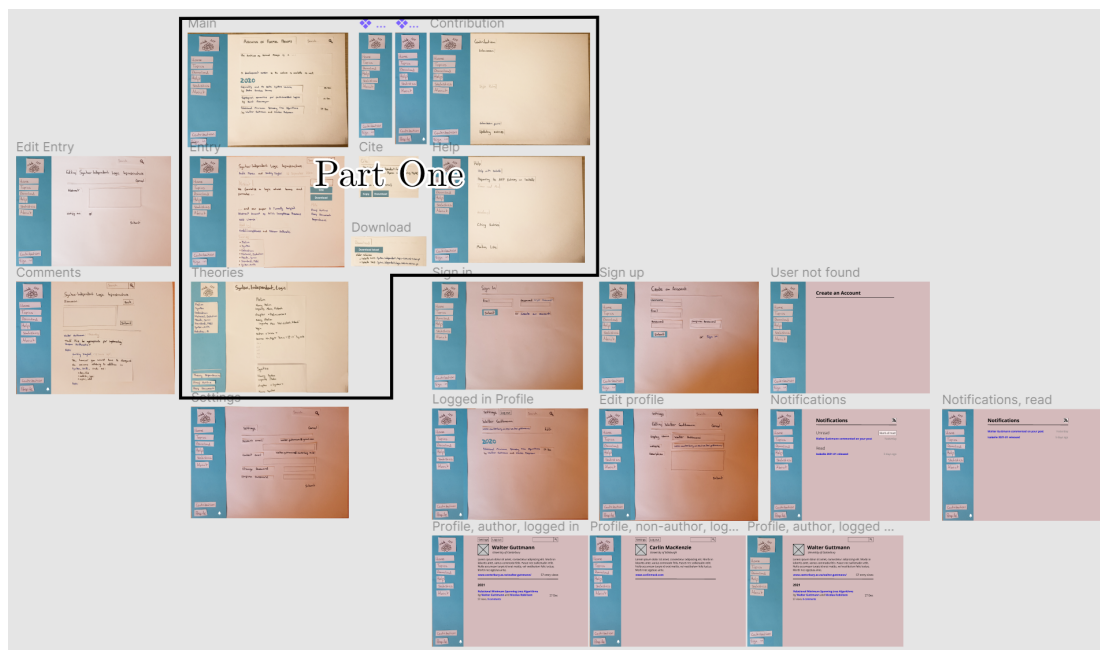


Figure 3.1: The final Figma artboard with the paper prototypes from Part One highlighted. The non-paper prototypes are the six prototypes in the bottom left with the same flat background

In the following sections we highlight the prototype and the resulting implementation of comments and notifications. All the paper prototypes from this Part are shown in Appendix A.

3.2.1 Comments

As the AFP is an archive, it is important that we have a clear demarcation between content that is being preserved and content that is non-curated, like user comments, so that users can have trust in the official information on the page. As such we considered two options for their placement. Inspired by Wikipedia's talk pages, we initially chose to have the comments on a separate page linked to from the entry (Figure 3.2). This

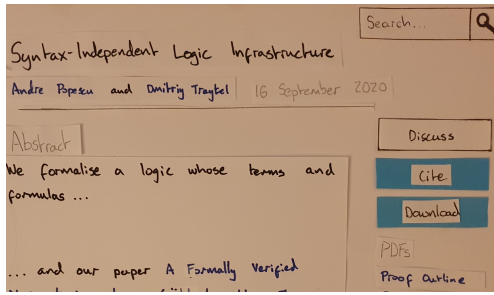


Figure 3.2

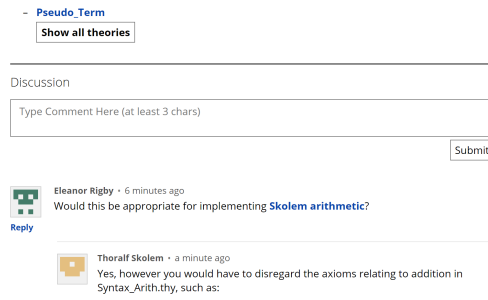


Figure 3.3

demarcates them the most, but we felt that most users would fail to notice the links to these pages. This would in turn render them mostly useless as the value comes from more people engaging with them. As such, we chose to place the comments at the bottom of the entry page with a horizontal line visually separating them from the entry (Figure 3.3).

3.2.2 Notifications



Figure 3.4: Prototype of the notifications page

Notifications

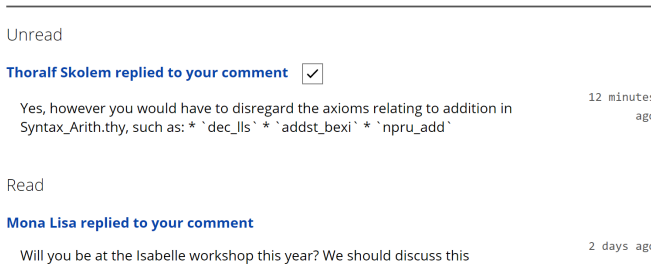


Figure 3.5: The notifications page

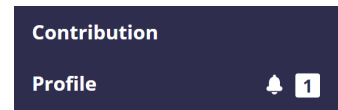


Figure 3.6: Notification link and unread indicator in menu



Figure 3.7: Unread indicator in tab bar

We place the link to the notifications near to the profile link in the bottom right as this is the area of the page which relates to the user. So that the unread counter is noticeable, we place it in a white rounded square next to the notification icon (Figure 3.6). White is chosen as we want the counter to not be distracting and draw the eye too much. We also add the count of unread notifications to the <title> (Figure 3.7). This means that the user will see that there is an unread notification even if they are not looking at the AFP tab at the time.

The notifications page lists all user notifications split into read and unread, with each section shown in reverse chronological order. When there are multiple unread notifications, a “Mark all read” button appears. The date is shown in a relative format (12 minutes ago, yesterday, etc.) and a preview of the reply is shown. The title of the notification is a link to the reply itself so that users do not have to remember which reply they clicked on.

3.3 Design Considerations

During the implementation of our features, we made many decisions outside of the prototyping context due to the number of designs being small. We list and justify these choices here.

3.3.1 Feedback Messages

Since their publication in 1993, Nielsen’s ten usability heuristics [28] have been widely used in the field of human computer interaction for evaluating the usability of systems [26]. The first heuristic is “visibility of system status”, which refers to the user’s ability to receive feedback on their actions. As we are adding interactive features to the AFP, we must add a mechanism for displaying adequate feedback.

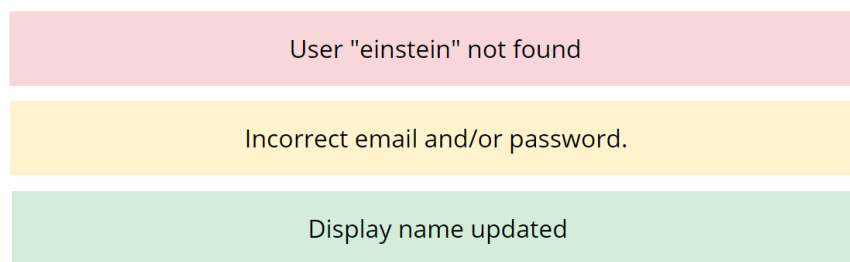


Figure 3.8: Feedback messages implemented for error, warning and success cases

We follow Bootstrap’s convention [29], and add messages for success, warn and error cases. Figure 3.8 demonstrates an example message for each case. As these are temporary information messages, the user would potentially spend more time dismissing the message than they would be on the page for. Therefore they can only be dismissed by reloading the page or navigating away.

3.3.2 Icons



Figure 3.9: Part one icons



Figure 3.10: Part two icons

In Part One of the project, we created the search, copy and download icons shown in Figure 3.9. We opted to round the corners of the icons so that they would match the

softer curves of our font, Open Sans. In this project we continued this design language and matched this style when creating the menu, notification bell, RSS, tick and pin icons (see Figure 3.10). For all icons, we made them as intuitive and simple as possible so that they were recognisable at small sizes.

3.3.3 Responsive Design

In Part One we made no specific considerations for larger and smaller devices. We now motivate a responsive design by considering where people are likely to be using the AFP. For example, people may check their notifications on their phone while they are away from their desk, and we want to ensure that they can still access the site comfortably.

We want to preserve the same style of navigation in the mobile design. As we want to hide it until the user needs it, we select to use a “hamburger” style menu. While it would be more ergonomic to have the navigation bar at the bottom of the screen, this is less traditional. It is more appropriate to keep this navigation element at the top of the screen where our users will expect it.

We add CSS media queries targeting two viewport widths: below 875px, for example a half desktop window and below 650px, for example mobile. The former is most important and signals the change to hide the side navigation and display the “hamburger” style menu instead. The latter is used for decreasing padding on the side of the screen so that content takes up the full width. Additionally all forms display in a single column at this width.

3.3.4 Web Feeds

Web feeds allow users to keep up to date with a site externally and so they help users that use them to be more engaged with that site. The most popular standard is Really Simple Syndication or RSS [2] (syndication being the ability for external readers to subscribe to a site). Unfortunately the standard is ambiguous and due to backwards compatibility requirements a lot of the issues cannot be fixed. Atom [20] is seen as an evolution of RSS and fixes issues with the grammar, internationalisation and time.

In the past it was popular to have a visible orange RSS icon on the page which would link users to the feed. However, the discontinuation of Google Reader has led to organisations placing less emphasis and even hiding their feed links. As such it is now inappropriate to surface this icon on the front page of the AFP, and instead we have opted to link to these feeds from the Help page. This is similar to the strategy that arXiv uses¹. We have placed a link to both feeds in the header of all author, topic and dependency pages.

3.4 Conclusion

This chapter presented the rationale behind our feature selection process, the design process, and the decisions behind the features we introduce. We discuss the implementation

¹<https://arxiv.org/help/rss>

of these features in the following chapter.

Chapter 4

Implementation

Our extension of the AFP features user accounts, comments, notifications, customisation and new traffic statistics. We maintain the static nature of the site and do not store any data with third-party services. In this chapter we describe how we achieved this and justify the design decisions that we made.

4.1 Server Architecture

Part One focused on the front-end and simply used the built-in Hugo web server and then eventually GitHub Pages to host the site. In this project we extend the AFP with a dynamic API and as such we need a suitable back-end. In this section we discuss our approach, the resulting server structure and finally how we hosted it publicly on the web.

4.1.1 Approach

The main constraint that we have in this project is adding dynamic features to a static site. With a dynamic site, users can request an arbitrary page to be created and served by the server if the format of the URL is correct. However with a static site, all URLs must correspond to a file on disk. We could generate files for each user profile, however this would quickly get unmanageable. As such we require an alternative mechanism.

We have chosen to use a modern version of the Asynchronous JavaScript and XML (AJAX) approach that was introduced in 2005 [14]. As shown in Figure 4.1, conventional websites serve HTML and CSS in response to HTTP requests. The AJAX model has the same first step but JavaScript is also returned and executed which requests the server again using `XMLHttpRequest`. Finally data is returned in XML format and JavaScript is used to manipulate the web page depending on the content of the data. Both models are possible with static and dynamic servers.

Instead of using `XMLHttpRequest` we use the simpler and newer `fetch` API which takes advantage of JavaScript promises. We use JSON as the data transfer language instead of XML as it directly maps to a data structure in JavaScript.

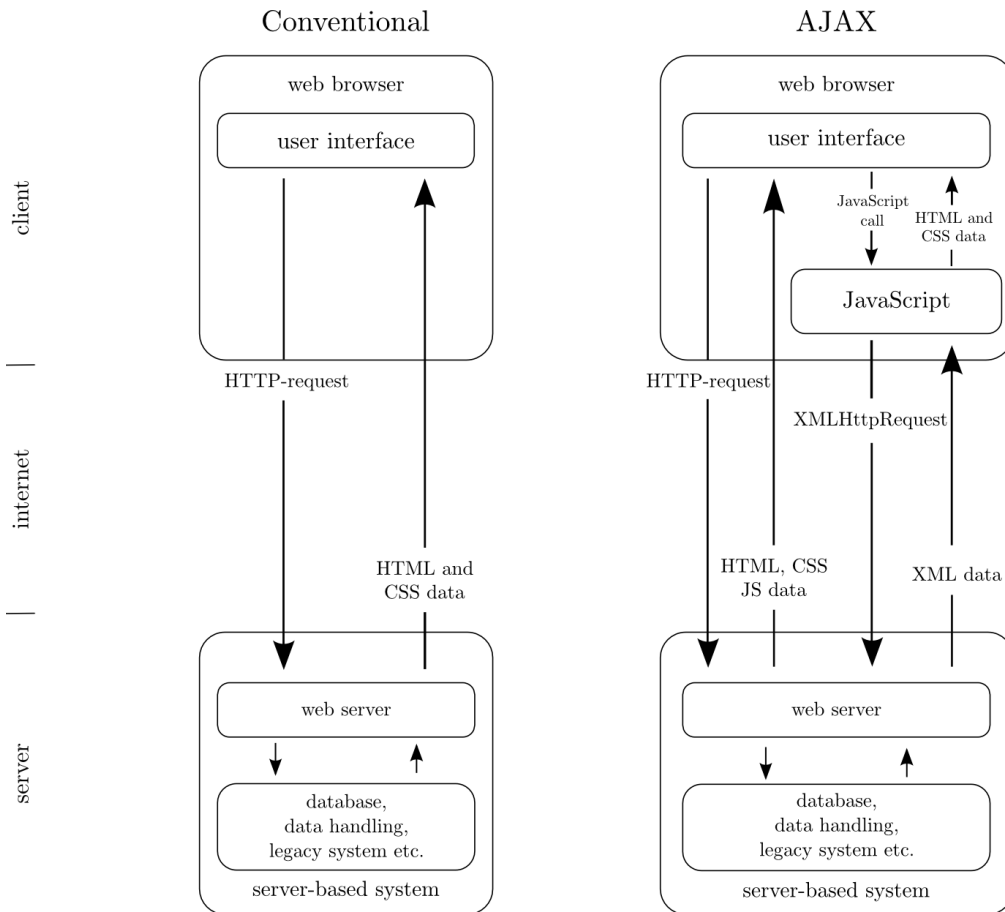


Figure 4.1: The conventional model of a web application versus the AJAX model which allows for asynchronous updating of the web page. Adapted from *Ajax-vergleich-en.svg* on Wikimedia Commons

As described, many sequential HTTP requests are made and the page is updated after page load. This is the same mechanism that is used for all features added by user accounts. Due to our notification system, we request the API on every page load when the user is logged in.

4.1.2 Structure

In this project we are preserving Hugo, the static site generator, which requires all pages to have a corresponding Markdown file associated with it. We keep Apache [11] as the primary web server as we want to preserve as much of the current site as possible. We expose a dynamic API in the server to provide the dynamic functionality.

As well as keeping with a file-centric approach, we did not want to introduce new programming languages or runtimes like PHP or Java. However, we did still need to add a server and decided upon a Node.js server [8] as we were using JavaScript on the front-end. Unfortunately, we also introduced a Python based server, and this decision is elaborated on in Section 4.6. The structure of the final server is shown in Figure 4.2

We expose the API by passing requests to certain paths to a respective port on the local-

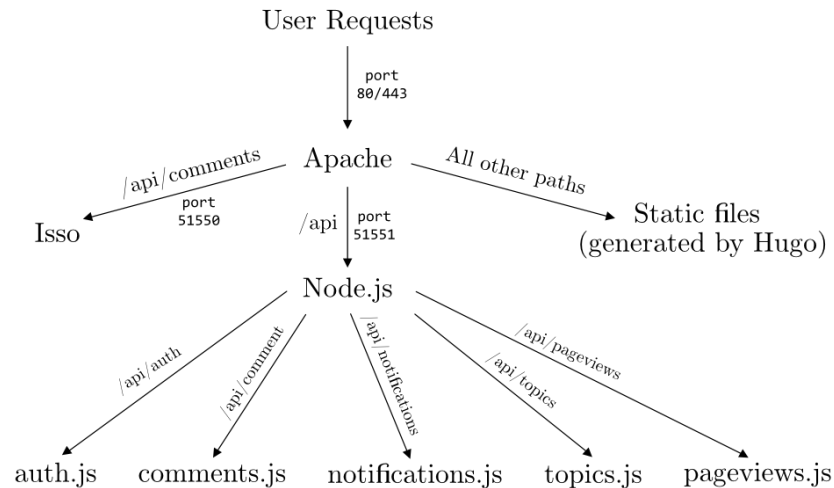


Figure 4.2: The web server structure

host. We chose ports based upon the Internet Assigned Numbers Authority procedure [7] which dictates the private use range is 49152-65535. We could use any available unused port in this range, but we chose 51550 for Issso (as 1550 looks like ISSO) and 51551 for Node. We use the `ProxyPass` directive as shown in Listing 4.1.2 to direct the traffic to each path to each server.

```

1 ProxyPass /api/comments http://localhost:51550
2 ProxyPass /api http://localhost:51551

```

Listing 4.1: Apache configuration for passing traffic to API

4.1.3 Hosting

The architecture described was hosted on Hetzner Cloud, a Virtual Private Server (VPS) provider based in Germany. It was chosen as it is much more affordable than the offerings by Microsoft Azure, Google Cloud or Amazon Web Services. The cheapest VPS tier was chosen which has 1 vCPU, 2GB RAM and 20GB of storage. It required 28 commands to install dependencies and start the servers (Appendix B). We use `tmux` [39] to keep the servers running in the background as one can attach and detach from a session at will. This provides easy access to starting, stopping and checking logs. HTTPS was enabled using the EFF's Certbot [9] which uses LetsEncrypt's certificates. Finally an A record was pointed to the public IP of the server in our DNS provider so that it could be accessed from a subdomain <https://afp.carlinmack.com/>.

4.2 Database

In Part One we stored all data as JSON in Markdown files. This was appropriate for the curated content, namely the Isabelle theories, we were dealing with. However we must now store user generated content. We could integrate it with Hugo—it would require an API, but requests would manipulate files on disk. For example, a user editing their

display name would cause the API to open their account's Markdown file, edit it, and write it to disk. This would then be rendered on the website by regenerating the site with Hugo. Editing text files adds significant complication, especially with user generated content. Additionally, regenerating the site creates lag until the page is updated as well as excess computation, re-rendering pages which have not changed. Instead we decided to keep all user generated content in a database.

There is a choice between relational (SQL) and non-relational (key-value store, document store and graph) databases. Relational databases are well-suited to our use-case and so it was a natural choice. In terms of which relational database to choose, we chose SQLite [34] as it is file-based, which matches Hugo; it does not require a server to accept queries, which makes deployment simpler; and can cope with at least 100k requests per day [35], which is well within the amount of traffic the AFP currently receives.

One downside of SQLite is that it is not fully SQL compliant, and as such many of the advanced SQL features are not available. As our needs are quite simple, this has not been an issue. Another potential weakness is that as there is no database management system—there is no such thing as a database administrator. As such, anyone who can read the file can also read all the data in the database. If this were to be integrated officially, a security audit would have to be performed to ensure that only trusted people were able to read the database.

4.2.1 Access

Initially when considering implementing the API we chose GraphQL [3] as our database query language. This is a wrapper for SQL that allows for queries to describe the structure that they would like to receive, demonstrated in Figure 4.3.

```

1 {orders {
2   id
3   productsList {
4     product {
5       name
6       price
7     }
8   }
9 }}
1  {"data": {
2   "orders": [{
3     "id": 1,
4     "productsList": [{
5       "product": {
6         "name": "orange",
7         "price": 1.5
8       },
9     ]},
10  ]}
11 }}
```

Figure 4.3: An example GraphQL query and response from Wikipedia. The client does not write SQL and instead sends the structure of the data they want to receive

We initially implemented a proof-of-concept Node.js GraphQL endpoint integrating with our SQLite database. As we developed other features, we did not find that adding a GraphQL wrapper would be beneficial for the few and simple queries we needed to perform. It seems that GraphQL is most beneficial when those developing the front-end do not know the details of the back-end. Additionally, it is beneficial for queries that

take advantage of a graph-like structure in the database, such as authorisation, however we did not implement group level authorisation in this project. Consequently, we removed the GraphQL endpoint and instead used the `sqlite3` node package to read and write to the database using SQL queries.

4.2.2 Schema

The database that we use has seven tables, four for comments, and one each for user accounts, statistics and topics. The four comments tables are all created and managed by Isso (Section 4.6), but we do alter the main comments table to add a `seen` flag. The schema for our tables are shown in Figure 4.4. As SQLite supports only five data types, namely `NULL`, `INTEGER`, `REAL`, `TEXT`, and `BLOB`, most fields are `TEXT`.

```
1 CREATE TABLE users (
2     username TEXT UNIQUE,
3     email TEXT UNIQUE,
4     hashed_password BLOB,
5     salt BLOB,
6     name TEXT,
7     image TEXT,
8     affiliation TEXT,
9     website TEXT,
10    description TEXT
11 );

1 CREATE TABLE logs (
2     status INTEGER,
3     request_method TEXT,
4     request_url TEXT,
5     date TEXT
6 );

8 CREATE TABLE topics (
9     username TEXT,
10    topic TEXT
11 );
```

Figure 4.4: Schemas for the `users`, `logs` and `topics` tables

When it came to adding pictures to user profiles, we had a choice about whether to store these in the database as `BLOB`s or as `TEXT` file paths. According to tests performed by the developers of SQLite, it is better to store files external to the database if the size of the files is greater than 100kB [36]. As pictures are often larger than this, we store files on disk and the paths to them in the database. This makes management/moderation of the pictures easier and reduces the size of the database.

4.3 API

For our features, we require a back-end to communicate with the client. One option we considered at the start of the project was using PHP as the interface for adding dynamic content and responding to requests. This would be possible by adding PHP code to the Hugo templates which would allow Hugo to generate PHP pages. Apache would then execute the PHP when pages were requested and PHP would fill in the dynamic content. We did not choose this as we did not want to introduce a new language, and the complexities that it would bring to the project.

We instead chose Node.js as we would have JavaScript on both the front and back-end, hopefully making development easier. Routing is provided by Express.js as it is the most popular and common Node.js routing framework.

4.3.1 Routing

As described in Section 4.1, the Node.js API listens on port 51551 and is available at the `/api/` path. The available endpoints are:

`/api/auth` This is the most extensive path and has two GET paths and four POST paths. GET `signed-in` returns the authentication status and POST `signout` logs out the user. POST `signup` and POST `signin` use the credentials passed in the body to perform the relevant task. POST `updateSettings` allows for the user profile to be edited and POST `getUser` is used to return profile information for a given user. Described fully in Section 4.4.1.

`/api/pageviews` POST `/` returns the number of views and downloads for the list of entries provided in the body of the request. GET `all` returns the views and downloads aggregated by month and year for all entries. This provides the mechanism for Section 4.5.

`/api/notifications` The root endpoint returns all the notifications for the currently logged in user, GET `unread` returns the number of unread notifications and POST `read` changes the state for the notification IDs provided in the body. Described fully in Section 4.7.

`/api/topics` GET `user` returns the list of topics which the currently logged in user has pinned. POST `pin` and POST `unpin` change the state of the database, adding and removing topics per user. See Section 4.8 for details.

`/api/comment` Provides a single GET path `/api/comment/mostRecent` which returns the most recent comment id. This is to reduce server load and is described in Section 4.7.

4.3.1.1 Redirection

If a user is not logged in, then they can only comment by signing in or creating an account. By default they would be directed to their account page, but this would be frustrating if they are intending to make a comment. As such, support has been added for page forwarding. The page that should be directed to at the completion of the flow is appended to the URL with a query string of the form `?next=/entries/example`. This query string is then taken from the URL by JavaScript and placed into the form on a hidden element. This allows the back-end to receive the value of the query string and redirect to that page after the request is satisfied.

4.4 User Accounts

It is possible to have interactive features on the client-side without an account system, for example users could pin topics or post comments with a username attached. However

customisations like the former could not be synced across different browsers and sessions and identities, like in the latter case, cannot be claimed. As such, we add accounts so that we can enable both functionalities.

4.4.1 Authentication

Authentication is important to ensure that the user can only edit their profile, among other reasons. Currently many services provide authentication via OAuth which allows users to authenticate via a social account like Facebook, Google or GitHub. Additionally, there are Identification-as-a-Service providers which allow websites to hand-off the authentication mechanism entirely. While these are more convenient for the developer, they come with privacy risks as these social services are given information about the user base. As we want to preserve user data on-site, we choose to implement authentication ourselves.

We chose Passport.js [17] as the mechanism for authentication as it is the most popular Node.js authentication framework. While it specialises in providing authentication with many different platforms, as we are handling the storage of credentials ourselves we use its *local* strategy.

4.4.1.1 Authorisation

While there is authorisation which allows users to edit their settings, we did not implement admin or author authorisation as outlined in Section 3.1.

4.4.1.2 Creating an Account

As one of the primary purposes of creating a profile is to make a comment, we wanted to make creating an account to make a comment as simple as possible. As such, we present the minimum amount of input necessary on the “Create an Account” page — username, email, password and confirm password. Originally only an email and password were required, however we need a way to serve a user’s profile. It is inappropriate to surface the user’s email address and so a username is necessary. We could generate a username for the users, however these are unlikely to be memorable or informative when sharing a link.

4.4.2 Profile

In terms of user engagement, profiles enable users to express themselves and find more about others. As such, we extended the current author profiles with a picture, a description and an affiliation. Unfortunately as we chose not to implement email verification (see Section 3.1) we did not enable the ability to claim an author profile as a user, as it would be trivial to impersonate someone.

4.4.3 User URL

As Hugo requires a file to back each page of the website, if we were to give each user a URL we would have to create a text file with their details that could then be rendered by Hugo. If the user then wanted to change any of their details, we would have to edit the file on disk and re-render the site. We would then also need to consider if we check these changes into version control. As such, we chose to have a generic account page and display other user profiles via the query string portion of the URL.

4.5 Statistics

We felt that users would be more engaged with the AFP if they could see the effect that their entries were having. However we did not want to add tracking to the site as this has privacy implications and would add data management responsibilities. As such we generate statistics via server-side logging. We parse the Apache server logs using `apache_log_parser` [25] and insert them into the database using `sqlite3` from the Python standard library. As shown in Figure 4.4 we do not keep IP addresses in the database, only which pages were requested, status, request method and the date. This means that summary statistics can be queried without compromising user privacy.

Dominique Unruh

<https://www.ut.ee/~unruh/>

21 entry views

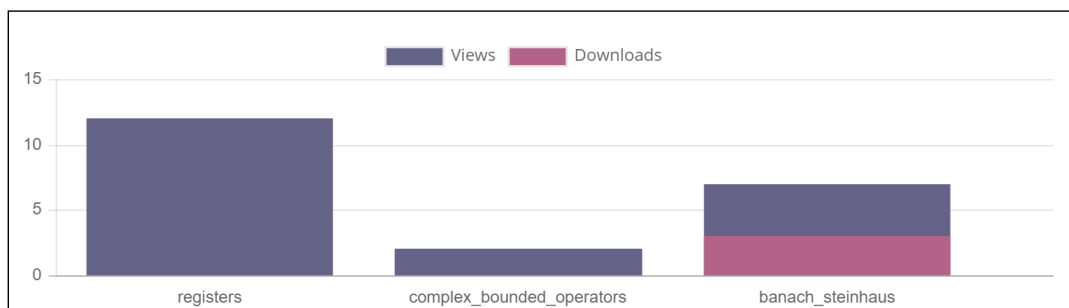


Figure 4.5: Chart showing views and downloads on an author's page

On all author pages we display an aggregate of the total number of entry views for their articles as well as individual views per entry. If they have three or more entries, we also display an interactive graph using `Chart.js` [38] (an inherited dependency from the current AFP) as shown in Figure 4.5. We display this as an overlapping bar chart, as the number of downloads will be less than or equal to the number of views. We display a similar interactive graph of the global page views and downloads on the statistics page (see Figure 4.6). Unlike the other charts on this page which are aggregated by year, this chart is aggregated by month and year as not enough time has passed to make a by year chart interesting.

Traffic received by AFP:

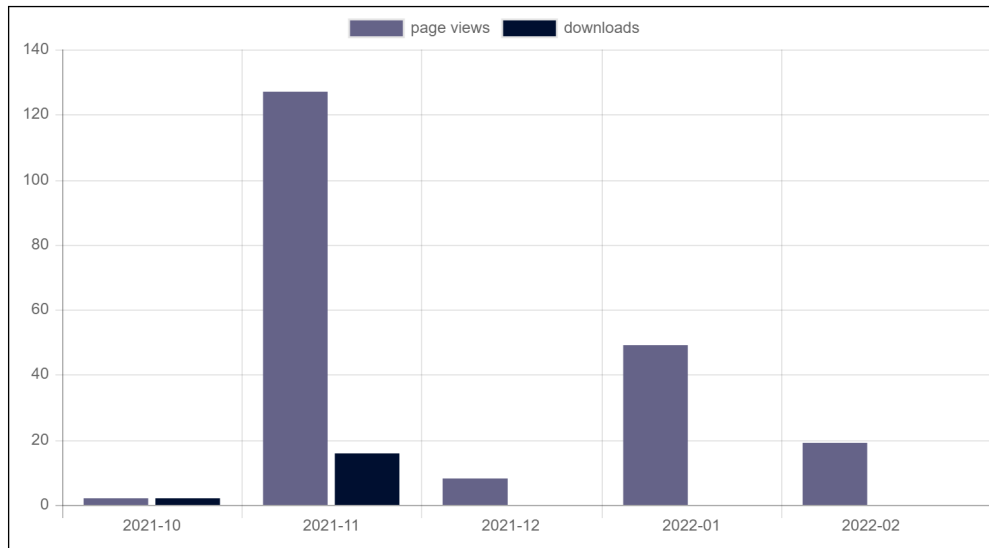


Figure 4.6: Chart showing views and downloads for the AFP as a whole

We update these statistics daily using a `cron` job which executes `logs.py` once a day. This ingests the logs created by Apache and inserts them into the database. For longevity of the project in production, a script could be created to aggregate the logs so that the database does not increase with the number of page views.

4.6 Commenting on Entries

One of the primary ways that we wish to promote community on the AFP is through the addition of comments. Not only do these allow space for people to ask questions and thank authors for their entries, but it also allows for the solutions to problems to be recorded next to the entry itself.

There are many options for providing comments on the Internet, currently the most popular being Disqus with over a billion monthly pageviews [31]. This service is easy to integrate for developers as it is free to use and only requires a single line of JavaScript to add it to the page. Hugo even ships with support for Disqus, allowing it to be enabled with an addition to the site's config file. Additionally, it is easy to use for the user as it provides sign-in with sites such as Facebook and Google. Unfortunately it is owned by Zeta Global which is a for-profit data-driven marketing company. Disqus serves to enrich their datasets which they sell to other companies.

Accordingly, there are many alternatives to Disqus with different features for different needs. We surveyed a number of available options in terms of the structural challenges they would pose to the project:

- Commento, Graph Comment, Hyvor Talk, Muut: These are paid services and we do not want to introduce any monetary costs.

- Valine: This cannot be self-hosted and we would like to keep our data on-site.
- IntenseDebate: Requires a WordPress website.
- Staticman: Requires Git, the AFP uses Mercurial.
- Utterances: Requires GitHub, the AFP uses a Mercurial repository service.
- Remark42, Talkyard, Cactus Comments: These require Docker which would be a significant introduction to this project.
- Cusdis, Mouthful: These do not provide Markdown support which makes them too simple for our needs. Markdown allows users to format their messages [16] which helps people to create messages which are easier to read.
- Isso: Requires Python.
- Schnack.js: This is almost the perfect solution, but it is no longer maintained.

As such we chose Isso [41] as the introduction of a Python server is the least impactful option. We installed Isso using `pip` and configured it to use our SQLite database. We then added the JavaScript file to all the entry pages using Hugo, and added the container where we wanted it to the entry pages according to our design in Section 3.2.1.

We had two main options to integrate it with our account system, editing Isso's code or building on top of it. We chose the latter as we would like to be able to upgrade Isso easily in the future.

Figure 4.7: Default Isso comment box

Figure 4.8: Our implemented comment box

By default, Isso presents users with an input as shown in Figure 4.7. We added a function to our `entries.js` script which hides them from the user but does not remove them as they need to be submitted with the form (Figure 4.8). Our function then checks if the user is logged in and if they are not the whole Isso form is replaced with a link to sign in. If they are logged in, the user's information is placed in the correct boxes.

4.7 Notifications

For a commenting system to be successful, users need to be notified of replies so that they have the opportunity to respond. As such we implemented notifications and a live updating notification counter.

4.7.1 Initial Implementation

Before figuring out the mechanism to integrate comment replies into the notifications, an announcement system was created so that the front-end could be implemented and

demonstrated. This was a simple table that stored a message, URL, date, user and a seen boolean. It was thought that an interface could be created for an admin to make an announcement, such as for the release of a new version of Isabelle, which would create a message for all users. However as we chose not to add authorisation (see Section 4.4.1.1), we ended up scrapping this table and functionality.

4.7.2 Comment Replies

Before any integration was done with Isso, the development documentation was reviewed to ensure that there was no preferable mechanism. By default, Isso has support for email notifications, however we have chosen to stay away from email in this project. Additionally in this review we found that there is a pull request to add support for webhooks¹ which would allow for direct integration. Alas, as neither option suits us, we build our notifications on top of Isso.

1	CREATE TABLE comments (1	
2	id INTEGER PRIMARY KEY	2	36
3	parent INTEGER,	3	35
4	text VARCHAR	4	Agreed!
5	author VARCHAR	5	Carlin MacKenzie
6	email VARCHAR	6	carlin.mackenzie@gmail.com
7	website VARCHAR,	7	https://afp.carlinmack.com/account
8);		/?user=carlin

Figure 4.9: Left: `comments` schema. Right: Example record. Only relevant columns are shown

Isso stores comments in the `comments` table as shown in Figure 4.9. The `parent` field stores the ID of the comment which it is replying to. We can therefore find all the replies to the comments of a user by finding all the comments which have parent comments written by our user of interest. We cannot use the name field to search on, as names are not guaranteed to be unique. We also cannot use emails as we do not surface these to the user, so therefore we match on the website which includes the username. Finally to store the state of whether the notification has been read, we add a `seen` boolean to Isso's `comments` table.

4.7.3 Unread Counter

To ensure that people are notified in a timely manner, it is important to add a live updating unread counter. It is therefore expected that the API will be pinged often by every single person that is currently on the AFP. Finally, as the SQL statement to request comments requires a subquery, it could potentially overload the server. As such, the first request of the user finds if there are any current unread messages. The response contains the number of unread messages, the number of comments the user has made and the most recent comment ID. If the user has not made any comments yet, the script stops. Otherwise, the `/api/comment/mostRecent` endpoint is queried repeatedly after a one

¹<https://github.com/posativ/isso/pull/724>

second delay. If there has been a comment then the unread endpoint is requested again and the cycle continues. If there is now an unread message the counter is displayed or updated, and the cycle continues. If the number of unread messages is greater than nine, a “+” is displayed instead so that it does not create layout problems.

4.8 Customising Topics

In their literature review, Garrett et al. [13] identified customisation as a feature which increases user engagement. Such customisation is not possible with our previous fully static site as all users are shown the same content. We chose topic pinning as the customisation to add as the topic page is long and dense, meaning that it can be difficult to find what one is looking for. In addition to this, we imagined that we could publicly display the topics that a user has pinned on their profile, as a sort of “Interested in”. However we decided against this as it would be an unexpected side effect, difficult to communicate that pinning and the topics displayed on their profile were linked and potentially unwanted by the user.

Topics

Pinned topics

Logic / Set theory

[Ordinal](#) [Ordinals_and_Cardinals](#) [HereditarilyFinite](#) [ZFC_in_HOL](#) [Forcing](#) [Recursion-Addition](#) [Ordinal_Partitions](#) [Delta_System_Lemma](#) [CZH_Foundations](#)

Computer science / Networks

[IP_Addresses](#) [Simple_Firewall](#) [Routing](#) [Iptables_Semantics](#) [LOFT](#) [UPF_Firewall](#)

Computer science

Algorithms

[MuchAdoAboutTwo](#) [SATSolverVerification](#) [Efficient-Mergesort](#) [Selection_Heap_Sort](#) [Boolean_Expression_Checkers](#) [Imperative_Insertion_Sort](#) [TortoiseHare](#) [Formal_SSA](#) [CYK](#)

Figure 4.10: The topics page with two pinned topics. Sections can be moved with the pin and unpin buttons

We add “pin” buttons to all second level headings (for example, Computer Science/Networks or Logic/Set Theory) as shown in Figure 4.10. We chose to have a first level “Pinned topics” section rather than moving the first level sections around as this would cause duplication of headings which may confuse the user. The unpin button is the same as the pin button, but with a score through it to depict that it is the opposite effect.

When clicked, the pin button triggers a POST request to the server which adds a row to the `topics` table with their username and the topic. The unpin button works similarly. When we display the page, we send a GET request to receive all the topics which the user has previously pinned. This mechanism is generic and could be used for other customisations on the site. For example, if we were to allow users to star favourite authors we could use a similar approach, only changing the HTML that is manipulated in the final step.

4.9 Feedback Messages

Passport.js has default support for displaying so-called “flash” messages to users, however this functionality is only available if one is using a fully dynamic back-end. As such we implement our own flash mechanism.

When the server reaches a state which the user would want confirmation or notice of, it sends a cookie with the HTTP request. This cookie will have the name `successMessage`, `warnMessage` or `errorMessage` and the value will be the message to display to the user. On the front-end, we place a function which checks for the presence of such a cookie and then creates and displays it.

4.10 Web Feeds

Hugo provides an RSS feed template by default but it is not valid RSS. As such, the RSS feed was corrected and the Atom feed was created. These were then enabled for the home page and all author, topic and dependency pages (known as *taxonomy term* pages in Hugo). We added a small dropdown to the header of these pages using the recognisable RSS icon as the label which expands to a list of the feeds available when it is clicked. A new section on the Help page was written explaining these feeds and how to access them. We also link to this information in the dropdown as we recognise that this technology is not as familiar as it once was.

4.11 Error Handling

In terms of the web servers, the Apache server must always run for the website to function. If the Isso server is down, then the comments will not be rendered but the user will not be notified. If the Node.js server is down, then a flash message will be displayed noting that the API is down.

If a URL that does not exist is requested, Apache will show a 404 page which is styled the same as the rest of the AFP and the user can either search or use the menu to navigate away.

If JavaScript is disabled, then the site degrades gracefully. For example using the `<noscript>` HTML tag, a “functionality does not work” message is displayed on forms and a “content will not display” message is placed where dynamic content should be. Additionally we serve forms with disabled inputs by default, and then enable them with JavaScript. This means that a user cannot accidentally submit a “Create an account” form and not be able to use that account. These considerations mean that the website is still functional and users are aware of what is missing.

4.11.1 Redundancy

As users can disable JavaScript we need to have server-side validation on all requests. We also need to have validation on the client-side to improve user experience as users can be notified of invalid input before sending the form.

4.12 Privacy

As the AFP is a project of both UK and European universities, we are bound by the General Data Protection Regulation (GDPR) [5]. GDPR is a directive which dictates how organisations handle users' data, especially in regards to privacy, security and control. We store user data in two places, for our account system and in our comments system, Isso.

For each comment that is made, Isso stores an IP address, name (which could be their username), email and website (which points to their profile on the AFP). The IP address is pseudo-anonymised as the last byte is dropped and is stored for basic spam prevention reasons (rate-limiting number of comments made). The name and email are both volunteered by the user and can be anything as these are not verified.

In terms of users requesting their information, this would be possible via a simple SQL query. Deletion of information would be a little trickier. Deleting all replies by a user is easy and not problematic. Deleting comments that are at the top level however results in all children comments being deleted. In this case, it would be best to blank the comment by replacing all fields with standard values indicating that they have been deleted. The SQL queries to request and delete information have not been created as part of this project.

Unfortunately security of the data storage is lacking with our implementation which is a violation of GDPR which demands that user data be encrypted. SQLite does not support access control or encryption by default. In production we could remedy this by using the SQLite Encryption Extension however careful consideration would be needed for key management.

4.12.1 Cookies

Cookies are only mentioned once in the GDPR, in which it states that cookies that are used to identify users are personal information and should be treated accordingly. The policy which relates to cookie banners is the ePrivacy Directive [4]. Cookies which are strictly necessary, i.e., the user is requesting a behaviour or if the site would not function without it, do not require a banner to be shown.

We use cookies in three places. First, our accounts system stores a session cookie to keep the user logged in, which is necessary otherwise users would have to send authentication details with each request. Second, Isso stores an encrypted cookie on the user's device to enable them to edit comments. Third, we store cookies to display feedback messages (see Section 4.9).

4.13 Conclusion

This chapter presented the implementation of the new back-end of the AFP and the features that were built on top of it such as user accounts, comments, notifications and statistics. In the next chapter we evaluate whether users find this to be an improvement.

Chapter 5

Evaluation

As our project concerns user engagement, it can only be evaluated by the target audience itself. Along with a user evaluation, we also evaluate the project technically, critically discussing the approach that we chose.

5.1 User Evaluation

In Part One we performed three formal user evaluations of the AFP, two before the start of the project and one after implementation. As we have this previous evaluation as a baseline, we chose to only evaluate this project after implementation.

5.1.1 Design

So that we can compare our results with the evaluation from Part One, we use it as our baseline, keeping the structure of the evaluation the same: a think aloud evaluation, followed by a multiple-choice survey and finally a long-form interview about areas of interest. The final script that was used during the evaluation can be found in Appendix D.3, along with the participant information sheet (Appendix D.1) and consent form (Appendix D.2).

One structural change that was made to the evaluation was that the order of tasks was changed between runs. One group (participants 1 and 3) was asked to create an account before creating a comment. The other group (participants 2 and 4) was asked to make a comment on the AFP, which implies creating an account. Both groups therefore perform the same tasks, but changing the order of the tasks allows us to compare the different paths to this goal. Additionally, each participant was asked to comment on a different entry so that they were not biased by the comments of previous participants.

Before performing the evaluation, the website and evaluation were tested with the fellow University of Edinburgh student who tested the version of the AFP from Part One of the project. They completed all tasks without difficulty and surfaced some final bugs which had been missed.

5.1.2 Results

As we wanted the participants to compare this extension with the redesign, we were constrained to asking the four people from the *Artificial Intelligence Modelling Lab* who participated in our evaluation in Part One. All four participants were available. The evaluations were not performed in the same order and so the feedback is not directly comparable between both parts. The evaluations were performed on the 4th, 7th and 14th of March and in order lasted 30 minutes, 16 minutes, 21 minutes and 15 minutes.

The think aloud section was successful, however there were some problems. During the first task with the first participant, there was a server error which crashed the page. This was very unfortunate and unexpected since a test run had been done beforehand. The error was promptly fixed as it was due to code that had erroneously been uncommented, and the think aloud continued as normal. During the “Add an affiliation” task with the third participant, there was an error as Chrome had auto-filled part of the form causing it to fail client-side validation. The participant eventually realised what had happened and cleared the input so that the form could be submitted. As we were on a call, Participant 2 chose to split his screen and have the call on one side and the website on the other half. Fortunately due to our work on making the site responsive (see Section 3.3.3), the subject was able to complete the evaluation in this configuration. They were the only participant to not see the unread notification indicator as it is hidden on the smaller viewport width.

1. For each of the following, please mark the box that best describes your reaction to the statement

[More Details](#)

Strongly Disagree Disagree Neutral Agree Strongly agree

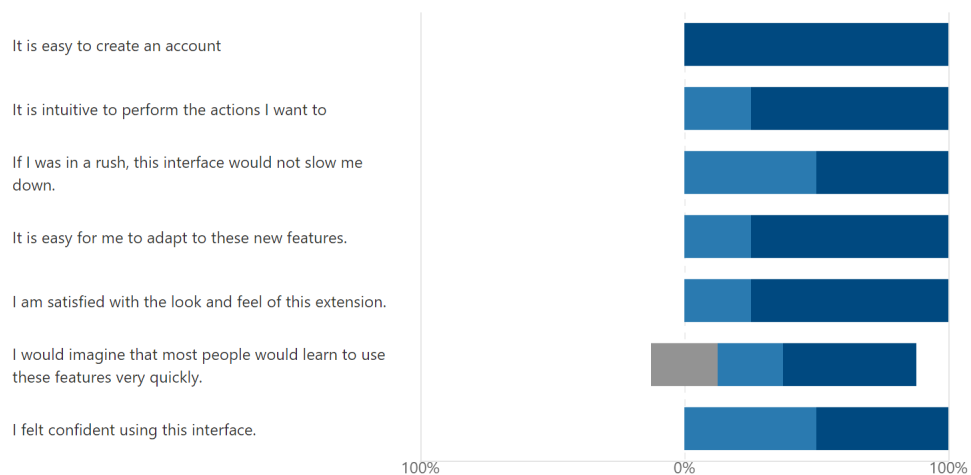


Figure 5.1: Short survey results

The results of the short survey can be seen in Figure 5.1. Unlike in Part One, there is one response which is not “Agree” or “Strongly Agree”. They explained that they felt that the user interface was only intuitive if you are used to similar design conventions from other websites. This did not prevent them from using the interface adequately, but that less experienced users might struggle. All other responses were positive.

Last, the participants answered the long-form questions.

Do these features appeal to you?

Participant 1 Overall they felt that these features were justified, but that they would appeal more with the addition of the author profiles. They suggested that the comments might be used for extra information which would be inappropriate for the abstract (links to a GitHub repository or a valuable StackExchange discussion) rather than long-form discussions.

Participant 2 They felt that they would rather use the Mailing List or Stack Exchange rather than the comments feature.

Participant 3 They liked the addition of comments as this provides a place to ask informal questions. They felt that they would email any detailed questions directly.

Participant 4 They felt that these features were good to have and that people would want to talk about the features on the site.

Would you be likely to create an account on the AFP?

Participant 1 They would create an account and possibly comment if they had a solution to a common problem that someone may have with an entry.

Participant 2 Even though they would not use the comments feature, they would create an account on the AFP. In general, creating an account on a website has a low overhead for them as they have a password manager and so prefers to log in if the option is there.

Participant 3 They would if they wanted to comment or if they were an author. Interestingly they thought that authors would have to create a profile, but this would be optional.

Participant 4 They would, but felt like they may only comment after they have submitted something themselves.

Is this an improvement over the redesigned AFP?

All participants felt that it was an improvement over the redesigned AFP from last year.

Participant 2 They felt that as the features are opt-in, the extension is either neutral or an improvement.

Participant 3 It's useful to be able to ask informal questions so they think that it is an improvement.

Are there any features that are lacking or missing?

Participant 1 As the AFP is an archive, they would not expect there to be social features. It might be nice to follow an author, however it is likely that one would see it on the mailing list anyway. Pinned comments would be useful so that authors could pin any notes to the top of the comments.

Participant 2 They suggested that it would be good to be able to search comments as it might be difficult to find information that in the comments of an entry. They also

would like to be able to see a history of their comments.

Participant 3 They felt that email notifications would be a good addition so that users do not have to rely on going to the website. In terms of comments, they asked if there was a length limit or perhaps if it is possible to hide replies from lengthy comment chains.

Participant 4 They did not think other social media features would be appropriate in this case. They would like some improvements to the profile as it was not immediately apparent that there were fields that they could fill in. They suggested having placeholder text like “No affiliation” that signalled that information could be added.

5.1.3 Conclusion

Similarly to Part One, all participants were able to complete the tasks and felt that it was an improvement. In contrast to Part One where all responses to the survey were positive, this time we had one neutral response. Also in Part One all improvements suggested were for the search feature, in this evaluation suggested improvements were across different areas.

In conclusion, the extension was well received and the features were seen as useful.

5.1.4 Resulting Changes

As a result of the evaluation we made changes to user profiles and the behaviour of notifications. For user profiles we add default values for the affiliation and profile picture. We also split the settings page and create an “Edit profile” page. This is simpler to implement than the original implementation and is simpler for the user to understand. Finally, to fix a bug with the settings page we disable `autofill`.

5.2 Technical Evaluation

The main challenge of this project was to introduce dynamic content onto a static site. As such, all dynamic content is fetched after page load and is not served by the primary web server. There are several limitations with this approach:

- While we can redirect users after page load and subsequent request to the server, we cannot prevent them from visiting any page with JavaScript disabled. This is not harmful, but it would be much nicer if parts of the website could be walled off completely. For example if admin pages were added, it would be much easier to figure out where to attack as a non-privileged user as the admin pages would be visible.
- In a similar way, we cannot enforce users to follow a multi-step flow through our app either. For example, we direct users to a page to fill in their name, affiliation, description, etc. however the user can click away and never come back to that page.

- Almost all of the extensions added by this project are unusable if JavaScript is disabled. If we had a dynamic server we could serve pages rendered with the dynamic content so that JavaScript was not required.
- As we are manipulating the DOM with JavaScript, the dynamic content added by JavaScript is brittle when changes are made to the structure of the page. This is fairly unavoidable with the approach we have chosen, however we can mitigate against this by ensuring the HTML and JavaScript match by versioning our source files. This works by appending a unique string to any resource's file name so that it is not loaded from the browser's cache [32].
- As our web server is static, we must rely on cookies to store state in the browser. If the user or their browser clears the cookies, a common thing to do for user privacy, all corresponding state will be removed from the site.
- For users with very slow or unstable connections, the page will take some time to become fully functional. Additionally, content loaded with JavaScript is not cached by the browser. This is because only HTTP requests are cached [10]. However this does mean that all data on the dynamic AFP pages are always fresh.
- We are constrained by what URLs we can use by Hugo. For example, user pages are accessed using the URL query string instead of a URL. This is not harmful in practice, but it would be nice if there were a way to map arbitrary content to URLs with Hugo. Due to its static nature this is not possible, so we would need to give Apache control over some of the URL space. This is possible, but we chose not to do it as it would add complexity.

Were this project to be implemented again, it would be good to investigate replacing Node.js with Python, potentially using the Flask framework. This would have complied with the principle of not introducing new languages, but potentially would be simpler. Node.js has added significant developer overhead and a rewrite was considered at one point to change from the CommonJS syntax to the Module based syntax .

In conclusion, our approach results in pages that are fast to load and robust. There are some shortcomings and complexity which come at the cost of satisfying our constraints. Nonetheless, this AJAX-style approach is still popular across the web, with most Web 2.0 websites changing the page after load (compare searching Google with JavaScript disabled).

5.3 Performance

In terms of serving HTML it is unlikely that the uptime of the website would degrade as the website is still fundamentally static. In fact, the HTML and CSS of the AFP are now smaller as they are minimised by Hugo. Apache only serves static files and directs traffic to the other API endpoints and so should remain performant compared to if it were rendering pages with PHP.

Node.js is optimised for I/O bound applications (rather than computationally bound) and so is a suitable choice for our API. If the Node.js API were to become unresponsive,

the AFP would remain available but users would not be able to sign in or comment—it would be read-only.

Isso is designed for small, self-hosted sites and so is not specifically optimised for performance. Even so, it is unlikely that enough comments would be made to cause performance issues.

Both Node.js and Isso rely on the SQLite database which can serve at least 100k requests per day [35] and so should be sufficient for the amount of traffic the AFP would realistically receive.

Even though we have not stress-tested our extension of the AFP with real traffic, due to the challenges of creating a realistic scenario, we can expect that it will be as performant as the current AFP and be able to handle the required load.

5.4 Maintenance

As this extension brings dynamic elements into play, we necessarily add maintenance complexity. The following summarises the technologies we have introduced and the amount of added complexity:

- *Apache* - While not new to the AFP, we have enabled two new modules. Apache is a slow-moving project and it is unlikely that it would break of its own accord or introduce backwards incompatible changes.
- *Isso* - Isso v0.5 was released in 2013 and is now on version v0.12.6.1 (last updated March 2022). It is fortunate that the project is still improved but it is unlikely it will add significant maintenance cost as it only receives a few updates per year.
- *Node.js* - Node is by far the biggest maintenance burden introduced in this project. While we can fix version numbers to ensure that things do not break, Node is notorious for needing frequent security upgrades.

5.5 Conclusion

In this chapter we have presented four evaluations of the AFP. First and foremost, our user evaluation was successful with users and their feedback has helped improve the final product. We did present some of the challenges with this approach, however they do not impact on the user experience. In terms of maintenance, this extension necessarily introduces complexity with its dynamic features. In the final chapter we conclude the project as a whole, summarising our contribution before elaborating on areas of future development.

Chapter 6

Conclusion

In this project we have extended the AFP with social features to increase user engagement. Users can now ask questions on articles, express themselves on their profile and authors can see the popularity of their work with statistics. We evaluated this with a group of users who found this extension to be an improvement, and subsequently adjusted the site based upon their feedback.

This project was less straightforward than Part One as we were more constrained and less familiar with the approach. I learned a great deal about back-end development and the additional workload it adds to a project.

Finally, this project was presented at the *Honours Project Day* and the poster that was created for it can be found in Appendix E.

6.1 Suitability for Production

While the set of features we have implemented are complete and our extension as a whole is functional, there are some features which would be necessary before official integration.

Resilient authentication Currently when the Node server is restarted all state is lost about logged in users. This would add unnecessary friction in a production environment and would have to be remedied. It is likely the solution lies in better API management with Apache.

Author Profiles As authorisation was not implemented in this project, authors cannot have a profile associated with them. Security of the site would have to be improved and emails would have to be validated so that someone could not impersonate another author which would be destructive.

Administration As the AFP would now accept user generated content in comments, it would be necessary to add sufficient moderator tooling to ensure that content is acceptable. Isso provides an admin interface and so comment moderation should be

acceptable at low volumes of comments. Attention would be needed to ensure that only editors of the AFP could access the admin page.

6.2 Future Work

There are many suitable features which would continue to advance this project:

Improvements to Comments Our evaluation with users resulted in many suggestions for improvements to comments—pinning, searching, collapsing and viewing a history of their comments. Isso is still being updated and merges pull requests so it would be possible to contribute these features.

Additional Social Features There are many social features which could provide benefits to the AFP. For example, users could like entries, bookmark them, or follow authors. Users would need to be consulted to understand if they want these features, as it is possible that these would be unhelpful or unused in practice.

Decentralised Web In development of our web feeds, we realised that it might be possible to create an ActivityPub endpoint. This is the protocol that decentralised social networks such as Mastodon use, and would allow a member of any decentralised social media to subscribe to the AFP. This would be akin to automatically posting new AFP entries on the decentralised versions of Twitter, Facebook, Instagram, YouTube etc. but comes with the benefit that we do not need accounts and/or API keys for each of them. There is discussion of adding support to Hugo¹, and adding static site support to the ActivityPub standard², however at the moment we would need to add an API endpoint and cryptographically sign our posts. This is possible, however is quite complex compared to an RSS feed where one only needs to host an XML file.

6.3 Concluding Remarks

Over the two parts which comprise this report, we have recreated, redesigned, improved, and extended the Archive of Formal Proofs. Both parts were evaluated by users of the AFP and found to be successful. It is therefore exciting that the community at large will benefit from our work as Part One is being integrated into the official AFP.

¹<https://github.com/gohugoio/hugo/issues/8135>

²<https://github.com/w3c/activitypub/issues/310>

Bibliography

- [1] Alex Woodson. Wikipedia remains go-to site for online news. <https://www.reuters.com/article/us-media-wikipedia/wikipedia-remains-go-to-site-for-online-news-idUSN0819429120070708>, 2007. [Accessed 5-May-2020].
- [2] RSS Advisory Board. RSS 2.0 Specification (Version 2.0.11), March 2009. <https://www.rssboard.org/rss-specification>.
- [3] Lee Byron. GraphQL: A data query language, September 2015. <https://engineering.fb.com/2015/09/14/core-data/graphql-a-data-query-language/> Accessed: 2022-03-22.
- [4] European Commission. Privacy and electronic communications directive 2002, 2002.
- [5] European Commission. General data protection regulation, 2016.
- [6] Mercurial Community. Mercurial. <https://www.mercurial-scm.org>.
- [7] Michelle Cotton, Lars Eggert, Joseph D. Touch, Magnus Westerlund, and Stuart Cheshire. Internet assigned numbers authority (Iana) procedures for the management of the service name and transport protocol port number registry. Request for Comments RFC 6335, Internet Engineering Task Force, August 2011. <https://www.rfc-editor.org/rfc/rfc6335.html>.
- [8] Mathias Dahl. Node JS. JSConf 2009, 2009.
- [9] EFF. Certbot. <https://certbot-prod.eff.org/>.
- [10] Roy T. Fielding, Mark Nottingham, and Julian Reschke. Hypertext transfer protocol (HTTP/1.1): caching. Request for Comments RFC 7234, Internet Engineering Task Force, June 2014. Available at <https://datatracker.ietf.org/doc/rfc7234/>.
- [11] The Apache Software Foundation. The Apache HTTP Server Project. <https://httpd.apache.org/>.
- [12] Wikimedia Foundation. Wikipedia. https://en.wikipedia.org/wiki/Main_Page.
- [13] Renee Garrett, Jason Chiu, Ly Zhang, and Sean D. Young. A literature review: website design and user engagement. *Online journal of communication and media*

- technologies*, 6(3):1–14, July 2016. <https://www.ncbi.nlm.nih.gov/pmc/articles/PMC4974011/>.
- [14] Jesse James Garrett. Ajax: A new approach to web applications. *Adaptive Path*, 2005. <https://web.archive.org/web/20150910072359/http://adaptivepath.org/ideas/ajax-new-approach-web-applications/>.
- [15] Inc. GitHub. GitHub, 2022. <https://github.com/>.
- [16] John Gruber. Markdown Syntax Documentation. Available at <https://daringfireball.net/projects/markdown/syntax>, March 2004. Accessed: 2021-04-10.
- [17] Jared Hanson. Passport.js, December 2021. <https://github.com/jaredhanson/passport>.
- [18] Kevin Hartnett. Building the mathematical library of the future, October 2020. <https://www.wired.com/story/the-effort-to-build-the-mathematical-library-of-the-future/>.
- [19] Kevin Hartnett. Building the mathematical library of the future, October 2020. <https://www.quantamagazine.org/building-the-mathematical-library-of-the-future-20201001/>.
- [20] Bill de hÓra and Joe Gregorio. The Atom publishing protocol. Request for Comments RFC 5023, Internet Engineering Task Force, October 2007. <https://datatracker.ietf.org/doc/rfc5023/>.
- [21] StackExchange Inc. Stackexchange. <https://stackexchange.com>.
- [22] Inc. Kandra Labs. Zulip, 2022. <https://zulipchat.com/>.
- [23] Carlin MacKenzie. Developing a New Web Application for the Archive of Formal Proofs. MInf Project (Part 1) Report, School of Informatics, University of Edinburgh, 2021.
- [24] Carlin MacKenzie, Jacques Fleuriot, and James Vaughan. An evaluation of the Archive of Formal Proofs. Available at <https://arxiv.org/abs/2104.01052>, 2021.
- [25] Amanda McCann. apache-log-parser, March 2015. <https://github.com/amandasaurus/apache-log-parser>.
- [26] Jakob Nielsen. *Usability engineering*. Academic Press, Boston, 1993.
- [27] Jakob Nielsen. Feature richness and user engagement, August 2007. <https://www.nngroup.com/articles/feature-richness-and-user-engagement/>.
- [28] Jakob Nielsen. 10 usability heuristics for user interface design, November 2020. <https://www.nngroup.com/articles/ten-usability-heuristics/>.
- [29] Mark Otto and Jacob Thornton. Alerts, January 2012. <https://getbootstrap.com/2.3.2/components.html#alerts>.
- [30] The Git Project. Git. <http://git-scm.com>.

- [31] Steve Roy. What's Cooler Than a Billion Monthly Uniques?, May 2013. <https://blog.disqus.com/whats-cooler-than-a-billion-monthly-uniqes>.
- [32] Nathan Sebastian. Versioning CSS files to invalidate browser cache.
- [33] LLC Slack Technologies. Slack. <https://slack.com/>.
- [34] SQLite. About SQLite, 2000. Available at <https://www.sqlite.org/about.html>.
- [35] SQLite. Appropriate uses for SQLite, 2015. Available at <https://www.sqlite.org/whentouse.html> Accessed: 2022-03-22.
- [36] SQLite. Internal Versus External BLOBs, May 2017. Available at <https://www.sqlite.org/intern-v-extern-blob.html> Accessed: 2022-03-22.
- [37] The Coq Proof Assistant. Community.
- [38] Evert Timberg and contributors. Chart.js. <https://www.chartjs.org>.
- [39] tmux Community. tmux, March 2015. <https://github.com/tmux/tmux/wiki>.
- [40] Cornell University. arXiv. <https://arxiv.org>.
- [41] Martin Zimmermann. Isso, March 2022. <https://posativ.org/isso/docs/>.
- [42] Zulip. Case study: Lean theorem prover community, November 2021. <https://zulipchat.com/case-studies/lean/>.

Appendix A

Paper Prototypes

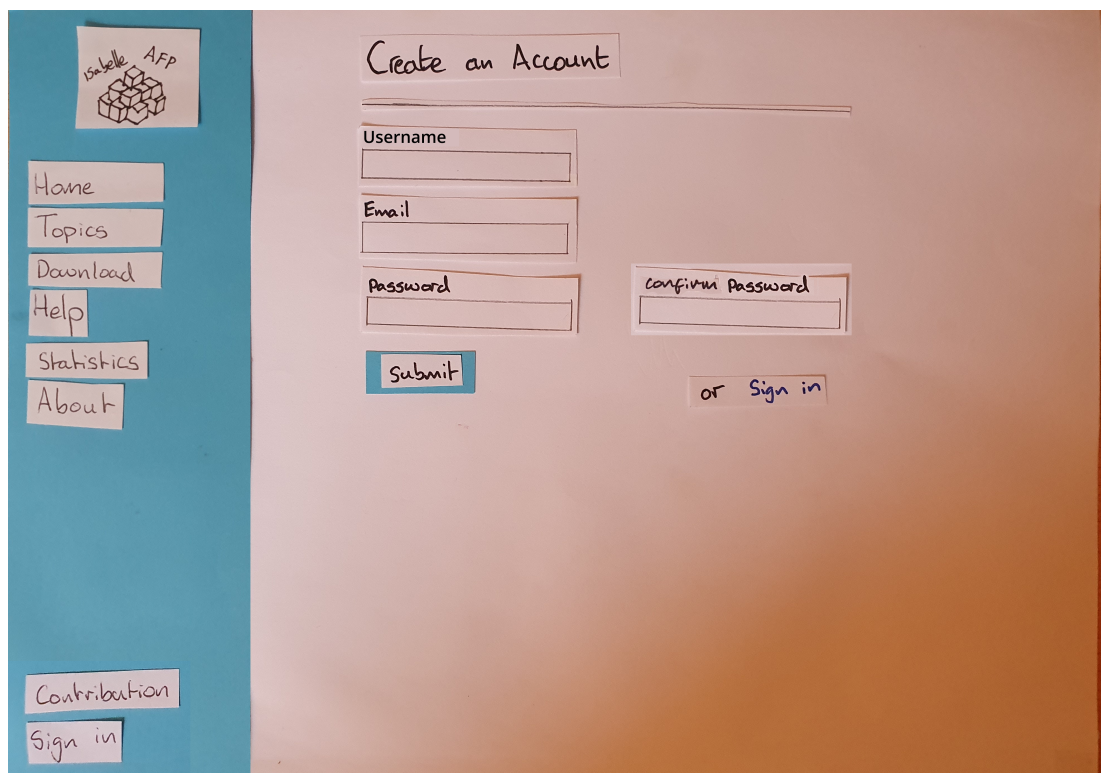


Figure A.1: The sign-up page. The link to this page is in the lower left

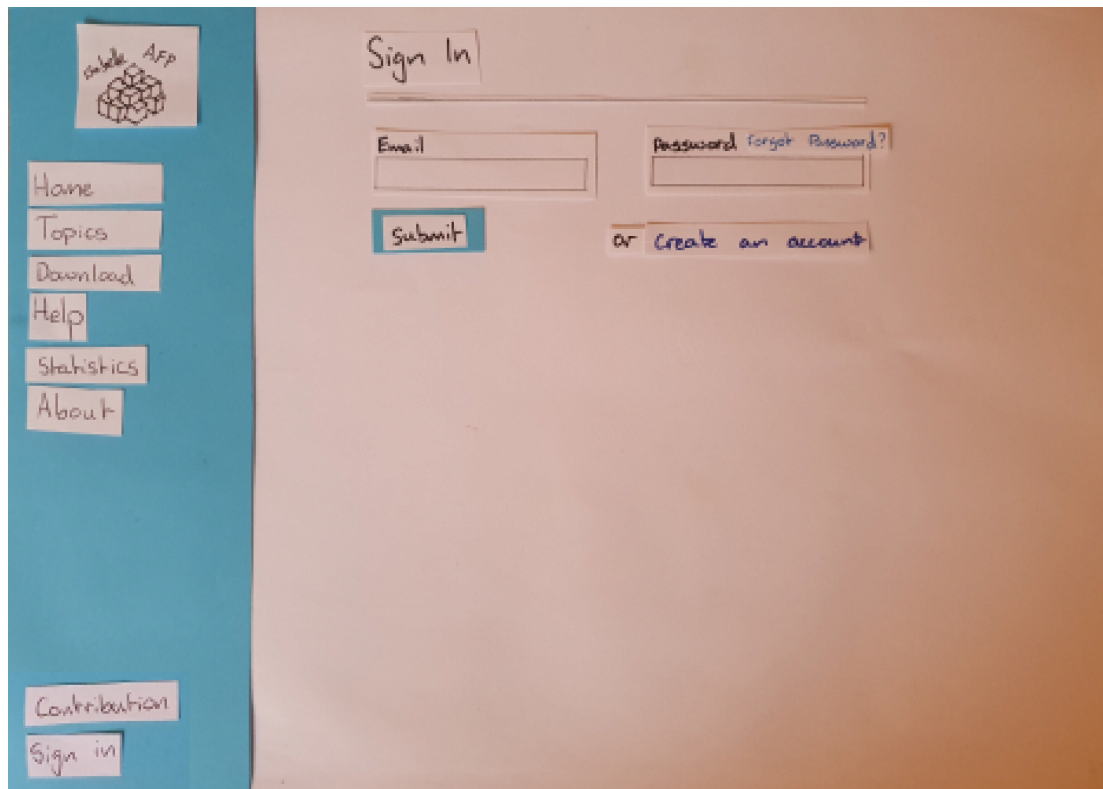


Figure A.2: The sign in page

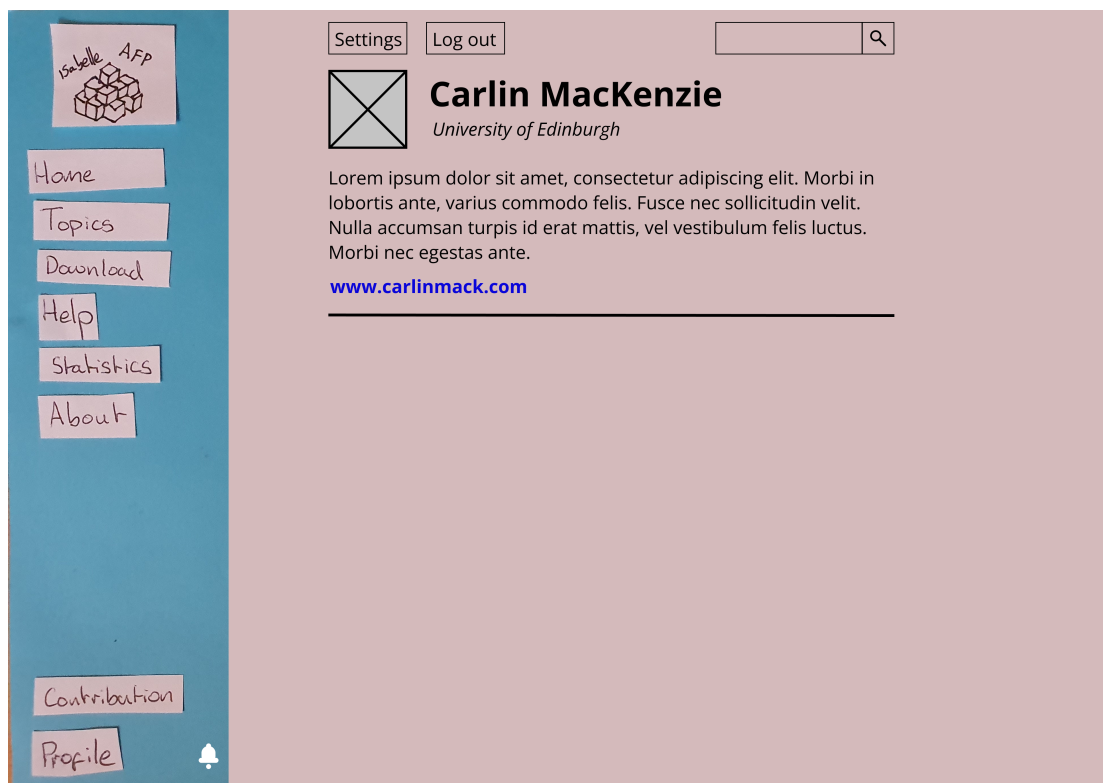


Figure A.3: An example user profile page

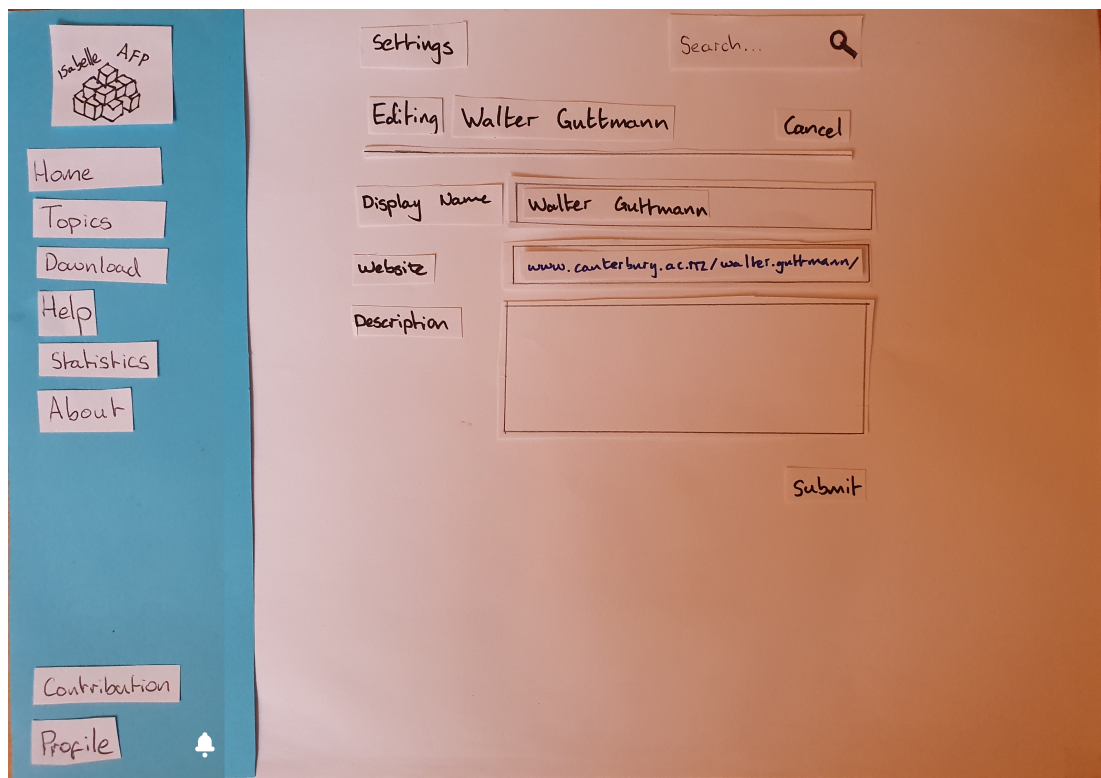


Figure A.4: The edit profile page

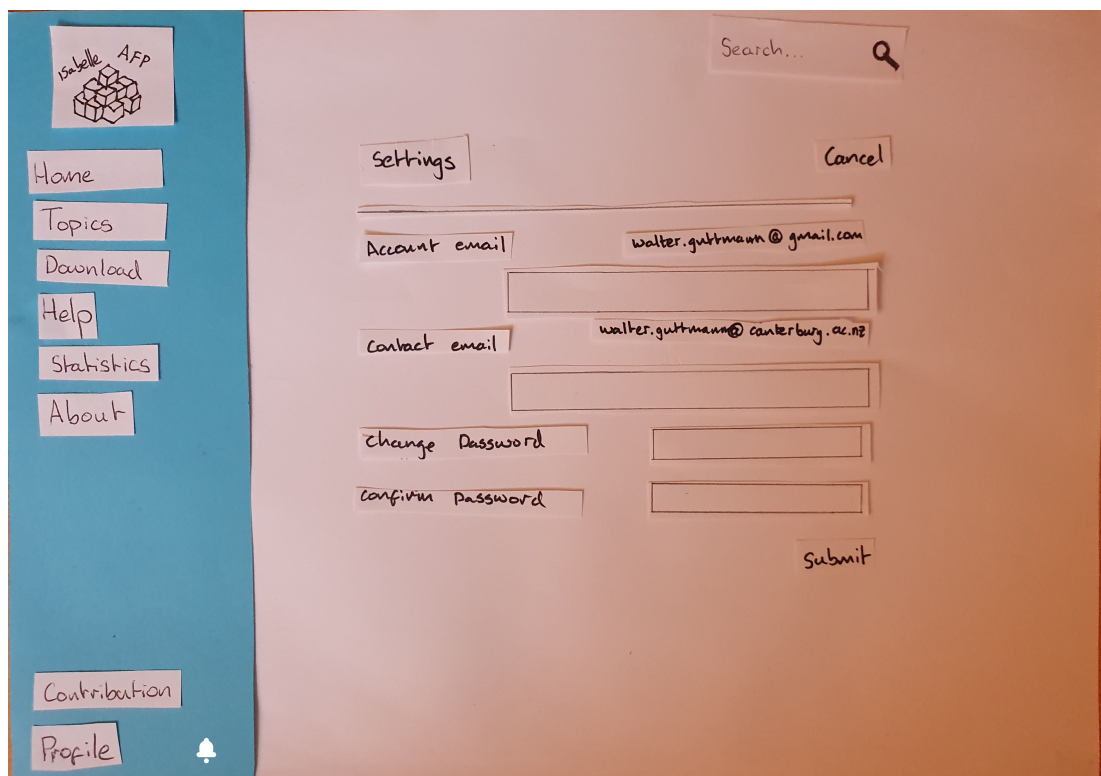


Figure A.5: The settings page. We combined this with the edit profile page in the implementation before user feedback told us they should be separate

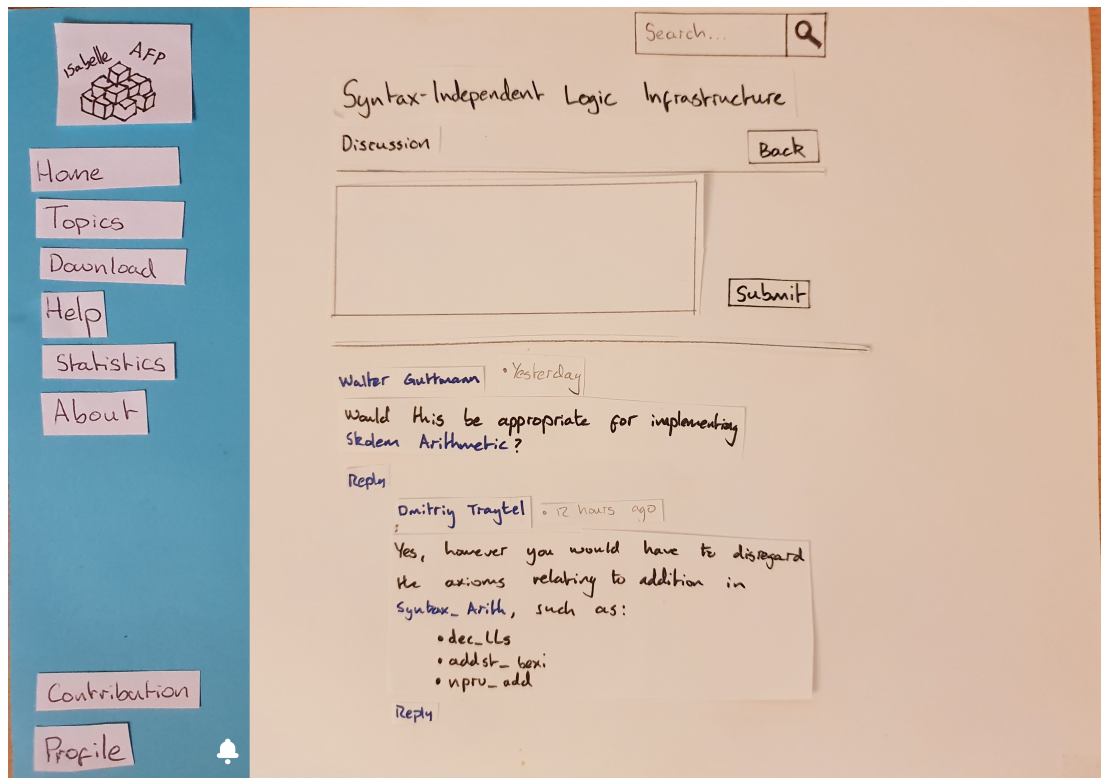


Figure A.6: A comments page. We decided to place this content at the bottom of the entry page (Section 3.2.1)

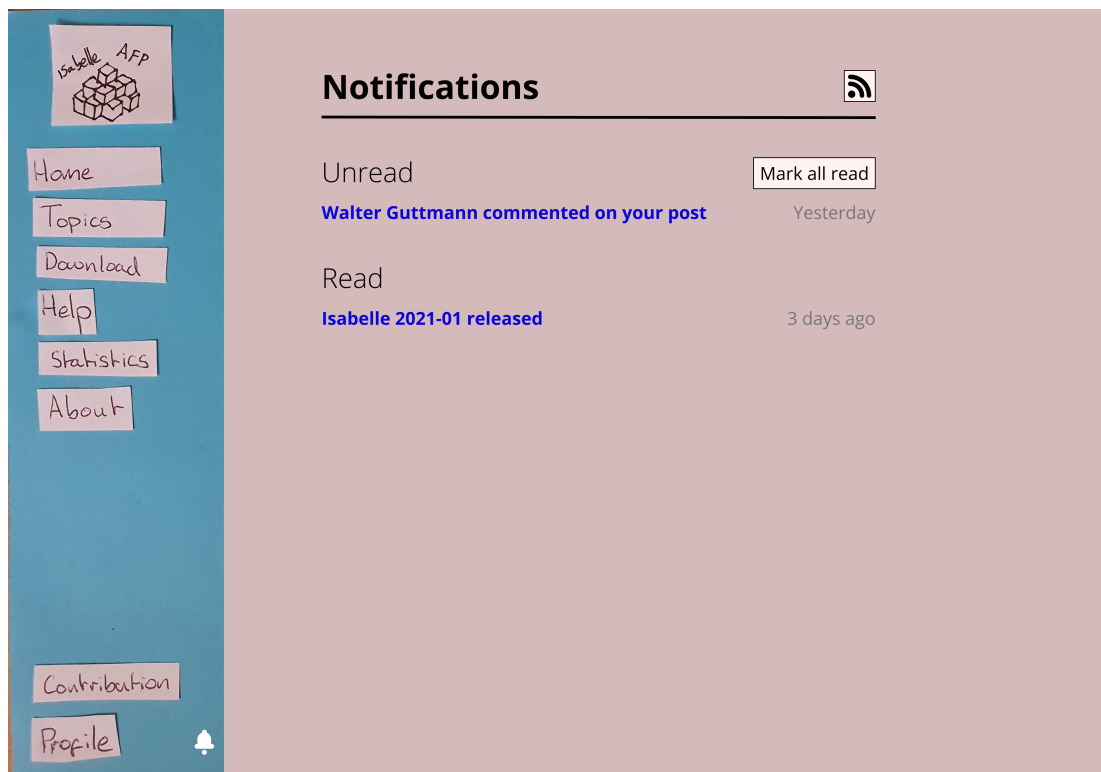


Figure A.7: The notifications page with unread notifications. In our implementation we placed a preview of the comment alongside the notification. We did not implement the RSS feed for notifications as it would need to be authenticated

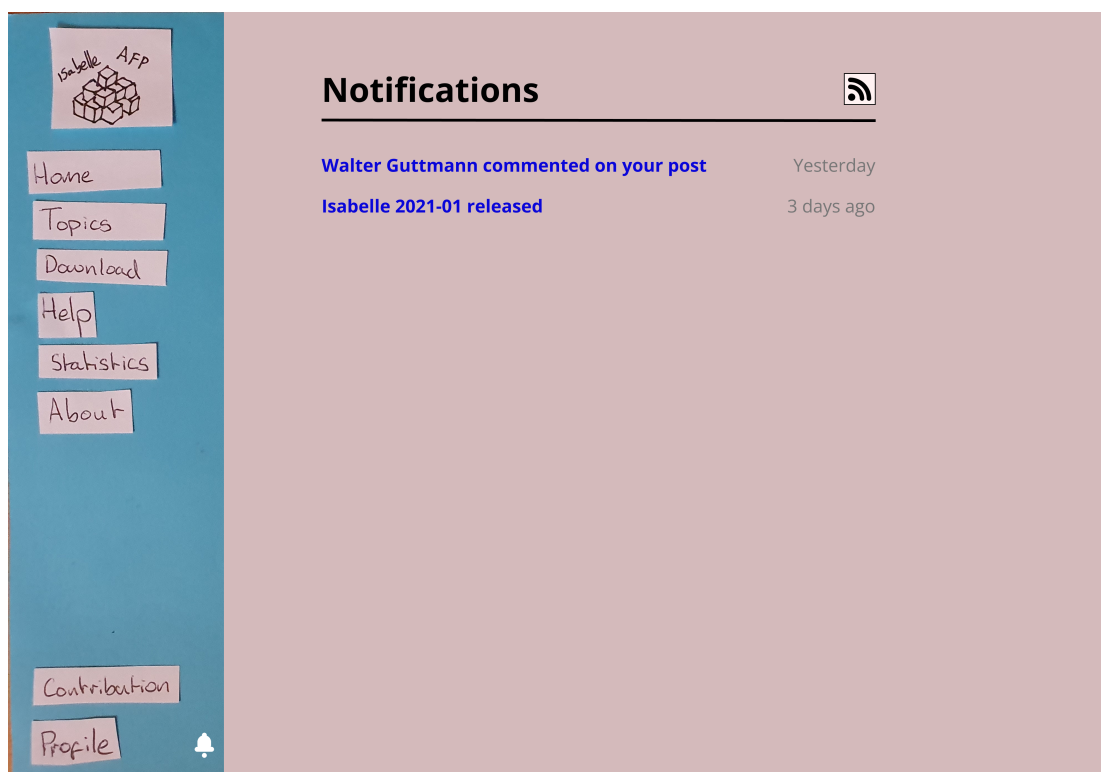


Figure A.8: The notifications page when all notifications are read

Appendix B

Commands to Host the Extended AFP

```
1 client: scp /etc/apache2/conf-enabled/node.conf root@65.21.55.61:/
   etc/apache2/conf-enabled/
2 client: ssh -i .ssh/id_ed25519 root@65.21.55.61
3
4 sudo apt update
5 sudo apt install apache2
6 sudo apt upgrade
7 git clone -b minf --depth 1 https://github.com/carlinmack/afp.git
8 cd afp/src/afp-devel/admin/hugo
9 wget -P /root/ https://github.com/gohugoio/hugo/releases/download/v0
   .92.2/hugo_extended_0.92.2_Linux-64bit.deb
10 sudo dpkg -i /root/hugo_extended_0.92.2_Linux-64bit.deb
11 sudo hugo --minify -d /var/www/html/
12 a2enmod proxy
13 apt install snapd
14 sudo snap install core; sudo snap refresh core
15 snap install --classic certbot
16 sudo certbot --apache
17 apt install isso
18 curl -o- https://raw.githubusercontent.com/nvm-sh/nvm/v0.39.1/
   install.sh | bash
19 nvm install --lts
20 cd ~/afp/src/afp-devel/admin/
21 wget -P /root/ https://bootstrap.pypa.io/get-pip.py
22 python3 /root/get-pip.py
23 python3 -m pip install -r requirements.txt
24 apt install sqlite3
25 mkdir /var/lib/sqlite
26 cd ~/afp/src/afp-devel/admin/hugo/api
27 npm install
28 npm start
29 cd ~/afp/src/afp-devel/admin/isso
30 isso -c isso.cfg run
31 service apache2 start
```

Appendix C

Screenshots of the Extended AFP

At time of submission, the redesigned AFP shown below can be viewed at <https://afp.carlinmack.com>

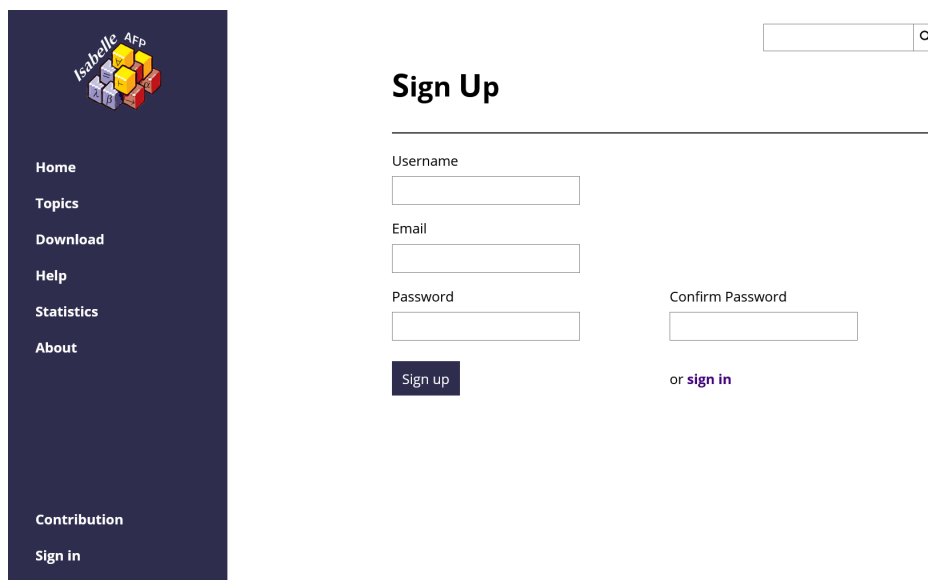


Figure C.1: The sign-up page

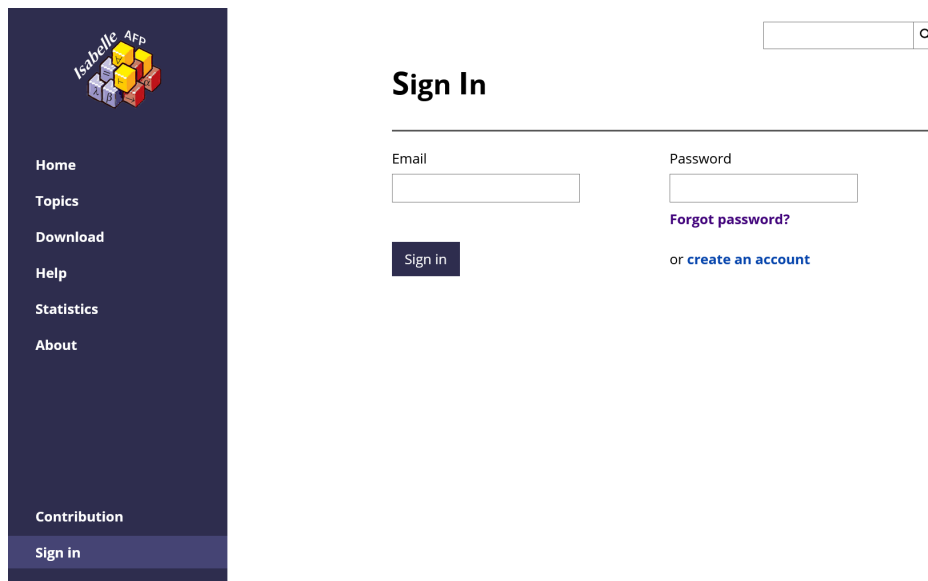


Figure C.2: The sign in page

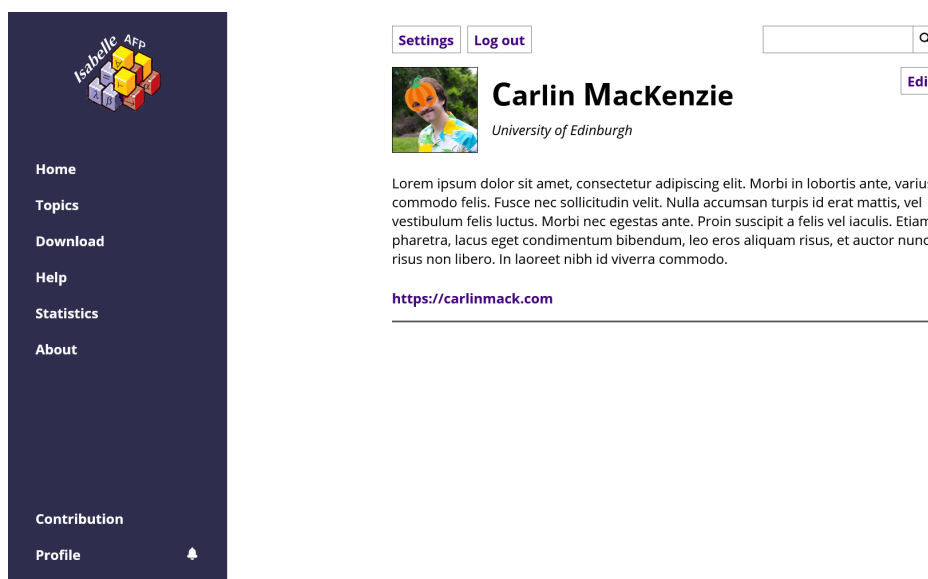


Figure C.3: An example user profile page

Isabelle AFP

Home
Topics
Download
Help
Statistics
About
Contribution
Profile

Search

Edit Profile

[Back](#)

Display Name

Affiliation

Website

Picture
 No file selected.

Description

Figure C.4: The edit profile page

Isabelle AFP

Home
Topics
Download
Help
Statistics
About
Contribution
Profile

Search

Settings

[Back](#)

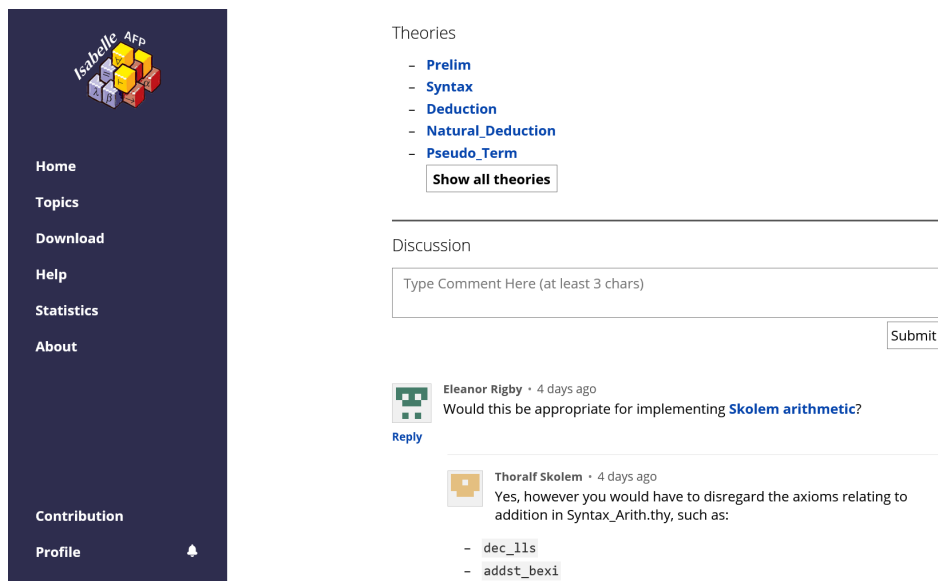
Account email

Current password

New password

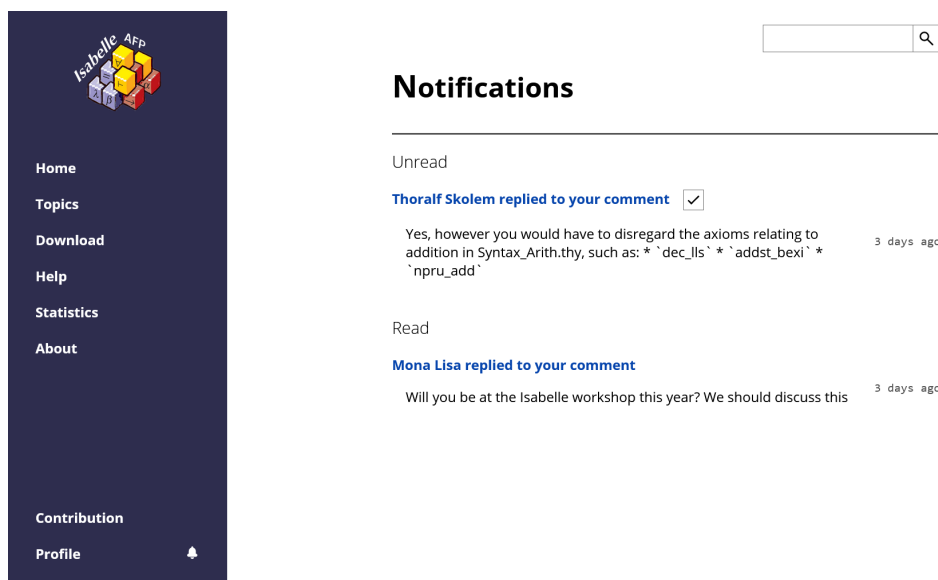
Confirm password

Figure C.5: The settings page. We combined this with the edit profile page in the implementation before user feedback told us they should be separate



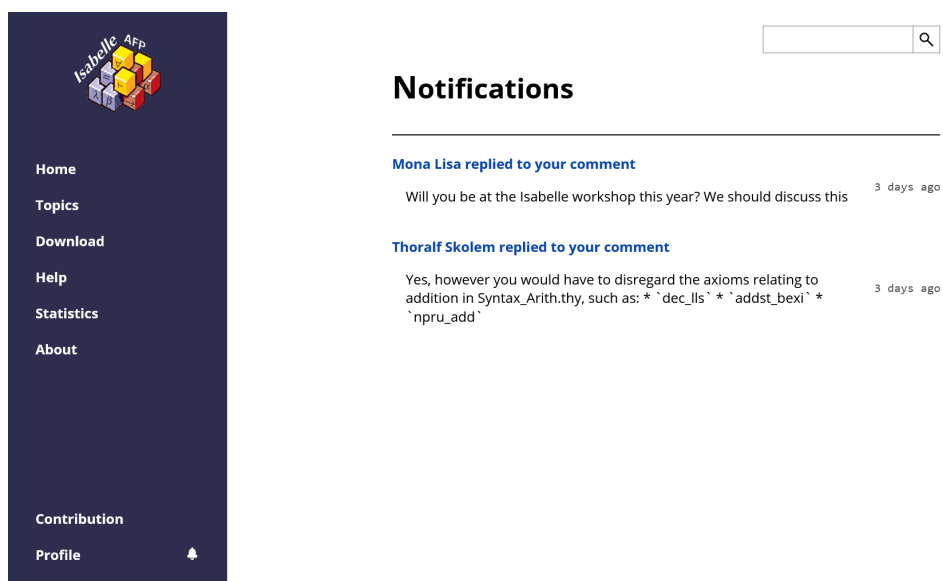
The screenshot shows the Isabelle AFP website interface. On the left is a dark blue sidebar with navigation links: Home, Topics, Download, Help, Statistics, About, Contribution, and Profile. The main content area is white and features a 'Theories' section with a list of links: Prelim, Syntax, Deduction, Natural_Deduction, and Pseudo_Term, followed by a 'Show all theories' button. Below this is a 'Discussion' section with a text input field containing the placeholder 'Type Comment Here (at least 3 chars)' and a 'Submit' button. A comment by Eleanor Rigby, dated 4 days ago, asks 'Would this be appropriate for implementing Skolem arithmetic?'. A reply by Thoralf Skolem, also dated 4 days ago, responds 'Yes, however you would have to disregard the axioms relating to addition in Syntax_Arith.thy, such as:' followed by a bulleted list of terms: 'dec_lls' and 'addst_bexi'.

Figure C.6: The comments section on an example entry)



The screenshot shows the Isabelle AFP website's notifications page. At the top right is a search bar. The main heading is 'Notifications'. Below it, there are two sections: 'Unread' and 'Read'. Under 'Unread', there is one notification: 'Thoralf Skolem replied to your comment' with a checkmark icon, dated '3 days ago'. The notification text includes a preview of the comment: 'Yes, however you would have to disregard the axioms relating to addition in Syntax_Arith.thy, such as: * `dec_lls` * `addst_bexi` * `npru_add`'. Under 'Read', there is one notification: 'Mona Lisa replied to your comment', dated '3 days ago', with the text 'Will you be at the Isabelle workshop this year? We should discuss this'.

Figure C.7: The notifications page with unread notifications. In our implementation we placed a preview of the comment alongside the notification. We did not implement the RSS feed for notifications as it would need to be authenticated



Isabelle AFP

- Home
- Topics
- Download
- Help
- Statistics
- About
- Contribution
- Profile

Notifications

[Mona Lisa replied to your comment](#) 3 days ago

Will you be at the Isabelle workshop this year? We should discuss this

[Thoralf Skolem replied to your comment](#) 3 days ago

Yes, however you would have to disregard the axioms relating to addition in `Syntax_Arith.thy`, such as: `* `dec_lls` * `addst_bexi` * `npru_add``

Figure C.8: The notifications page when all notifications are read

Appendix D

Evaluation

D.1 Participants' Information Sheet

Participant Information Sheet

Project title:	Evaluating an extension of the Archive of Formal Proofs
Principal investigator:	Carlin MacKenzie

Please take time to read the following information carefully. You should keep this page for your records.

Who are the researchers?

The research is being carried out as part of the Informatics Honours Project at the University of Edinburgh. This project is supervised by Jacques Fleuriot and James Vaughan. Today's study is designed and run by Carlin MacKenzie who is the primary researcher.

What is the purpose of the study?

The goal of the project as a whole is to design a new interface for the Archive of Formal Proofs and the purpose of this study is to evaluate a redesign of the AFP to understand if it meets the needs of users.

Why have I been asked to take part?

We are looking for people who have experience with Archive of Formal Proofs. You have been asked to participate because we believe that you have this type of experience.

Do I have to take part?

No – participation in this study is entirely up to you. You can withdraw from the study at any time, up until March 31 2022 without giving a reason. After this point, personal data will be deleted and anonymised data will be combined such that it is impossible to remove individual information from the analysis. Your rights will not be affected. If you wish to withdraw, contact the PI. We will keep copies of your original consent, and of your withdrawal request.

What will happen if I decide to take part?



You will be interacting with the redesigned website and the session should take about 30 minutes. We will ask you to do some normal activities such as finding entries and browsing the theory code. During the session we will ask you to share your screen with us so that we can watch you interact with the prototype. If you agree, we will video record the session so that we can review it in more detail later. Our goal is to understand if this design is successful, so seeing you interact with it will greatly help us understand where it supports you well and where it can be improved.

After interacting with the prototype, you will answer some multiple-choice questions and a couple long answer questions about the redesign.

Are there any risks associated with taking part?

There are no significant risks associated with participation.

Are there any benefits associated with taking part?

There are no direct benefits from taking part in this study other than the knowledge that you have helped us complete my honours project and possibly also help improve the live Archive of Formal Proofs site.

What will happen to the results of this study?

The results of this study may be summarised in published articles, reports and presentations. Quotes or key findings will be anonymized: We will remove any information that could, in our assessment, allow anyone to identify you. With your consent, information can also be used for future research. Your data may be archived for a maximum of 1 year. All potentially identifiable data including consent forms will be deleted within this timeframe if it has not already been deleted as part of anonymization.

Data protection and confidentiality.

Your data will be processed in accordance with Data Protection Law. All information collected about you will be kept strictly confidential. Your data will only be viewed by the research team: Carlin MacKenzie, Jacques Fleuriot and James Vaughan.

What are my data protection rights?



The University of Edinburgh is a Data Controller for the information you provide. You have the right to access information held about you. Your right of access can be exercised in accordance Data Protection Law. You also have other rights including rights of correction, erasure and objection. For more details, including the right to lodge a complaint with the Information Commissioner's Office, please visit www.ico.org.uk. Questions, comments and requests about your personal data can also be sent to the University Data Protection Officer at dpo@ed.ac.uk.

Who can I contact?

If you have any further questions about the study, please contact the lead researcher, Carlin MacKenzie <s1724780@ed.ac.uk>. Or the supervisors of the honours project, Jacques Fleuriot <jdf@inf.ed.ac.uk> and James Vaughan <s0952880@ed.ac.uk >.

If you wish to make a complaint about the study, please contact inf-ethics@inf.ed.ac.uk. When you contact us, please provide the study title and detail the nature of your complaint.

Updated information.

If the research project changes in any way, an updated Participant Information Sheet will be made available on <https://web.inf.ed.ac.uk/infweb/research/study-updates>.

Alternative formats.

To request this document in an alternative format, such as large print or on coloured paper, please contact Carlin MacKenzie <s1724780@ed.ac.uk>.

General information.

For general information about how we use your data, go to: edin.ac/privacy-research



D.2 Participants' Consent Form

Participant number: _____

Participant Consent Form

Project title:	Evaluating an extension of the Archive of Formal Proofs
Principal investigator (PI):	Carlin MacKenzie (s1724780@ed.ac.uk)

By participating in the study you agree that:

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

Please tick yes or no for each of these statements.

1.	I agree to being audio recorded.		
		Yes	No
2.	I agree to my screen being recorded by the researchers.		
		Yes	No
3.	I agree to take part in this study.		
		Yes	No

Name of person giving consent

Date

Signature (ok to type it)



THE UNIVERSITY of EDINBURGH
informatics

D.3 Script for the Evaluation

Hello, I'm Carlin and today we will be evaluating an extension of the Archive of Formal Proofs. Your participation today is purely voluntary, you may stop at any time.

Before we start, I just want to confirm that you've read the participation sheet and signed the consent form. If not, you can do that now. [After they have confirmed/signed] Is it okay for me to start recording the call now?

Similarly to last time, we'll first be doing a talk aloud evaluation. Do you remember and understand what I mean by talk aloud? [If yes, skip to "If you have any questions..."] So in this observation, I am interested in what you think about, as you perform the tasks you're asked to do. To do this, I am going to ask you to talk aloud as you work on the task.

What I mean by "talk aloud" is that I want you to tell me everything you are thinking from the first time you see the statement of the task till you finish the task. I would like you to talk aloud constantly from the time I give you the task till you have completed it. I do not want you to try and plan out what you say or try to explain to me what you are saying. Just act as if you were alone, speaking to yourself. It is most important that you keep talking and I will prompt you if you are silent for a long period of time. Do you understand what I want you to do?

Good. We'll start with a simple practice problem first. I will demonstrate by thinking aloud while I solve a simple problem: "How many pillows are there in my parents' house?" [Demonstrate thinking aloud.] Please verbalise like this as you are doing the tasks. If you have any questions, feel free to ask them and I will answer them after the session. Is this all clear?

First, I would like you to open a browser and go to the link which I will send in the chat. [When they have confirmed they have done so] Thank you, could you now share your screen?

<https://afp.carlinmack.com/>

I have prepared 5 [or 4] tasks for you to do which I'll send over Teams. For each one please read it aloud, complete it to the best of your ability and say "done" when you feel that you have completed the task. Lastly take your time, remember that I'm testing the interface, not you!

1. Create an account on the AFP using a dummy password.
2. Make a comment on the "Topology" entry and return to the home page
3. Add an affiliation to your profile [If they did already, Update your affiliation on your profile or Add a description to your profile]
4. Respond to my reply on your comment.
5. View my profile

OR

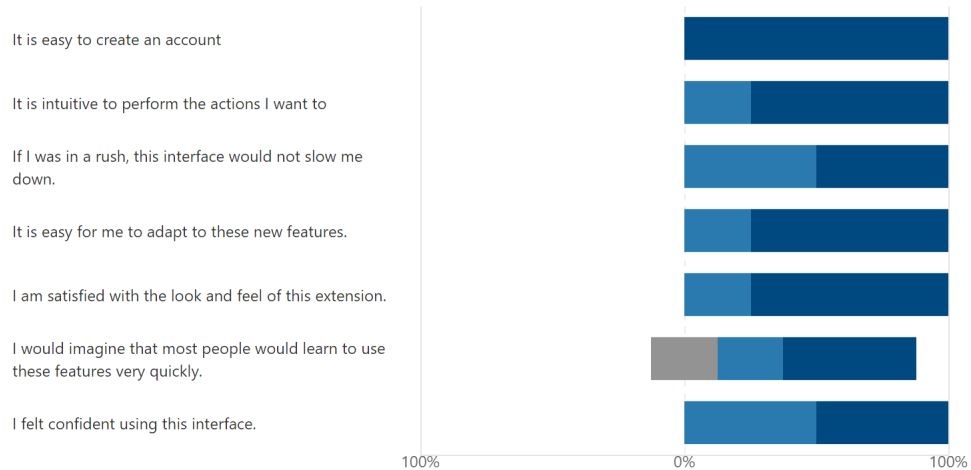
1. Make a comment on the “Completeness Theorem” entry and return to the home page
2. Add an affiliation to your profile [If they did already, Update your affiliation on your profile or Add a description to your profile]
3. Respond to my reply on your comment.
4. View my profile

Now that you have completed the tasks, I will send you a link to a survey which I would like you to answer. You can stop sharing your screen now, and please feel free to take your time and click around the website if you need a reminder. Let me know when you have completed it.

1. For each of the following, please mark the box that best describes your reaction to the statement

[More Details](#)

■ Strongly Disagree
 ■ Disagree
 ■ Neutral
 ■ Agree
 ■ Strongly agree



[Visit the survey summary page while they fill in and see if you can see their submission]

Optional: Before the final section, I'd like to hear a little bit about why you answered X for Y?

Lastly, I'd like you to answer some final open-ended questions.

1. Do these features appeal to you? How so/how not?
2. Would you be likely to create an account on the AFP?
3. Is this an improvement over the redesigned AFP? How so/how not?
4. Are there any features that are lacking or missing?

This is the end of experiment, thank you so much for your time, it was really appreciated.

Appendix E

Poster

The poster is titled "Extending the Archive of Formal Proofs" and features a dark blue header with the Isabelle AFP logo on the left and the University of Edinburgh logo on the right. The main content is divided into several sections:

- Constraints:** The Isabelle community favours a DIY approach that keeps user data on-site. Additionally, we want to keep the site as static as possible, meaning that we must request all dynamic content after page load.
- Implementation:** Database: SQLite is our chosen database as we wanted to preserve the file-oriented approach from last time. API: Node.js + Express provides the API for authentication, notifications and statistics.
- Evaluation:** To assess the success of our project, we created a mixed format evaluation which we ran with four members of the Artificial Intelligence Modelling Lab. Design: A think aloud user test, followed by a quantitative survey, followed by a long-form qualitative interview. Results: All participants were able to complete the tasks in the think aloud. All participants thought that it was an improvement over the redesign last year and that they would make an account on the AFP. They suggested many ideas for new features are improvements. Resulting Changes: As a result of the evaluation we plan to make adjustments to the behaviour of notifications and the display of profiles with incomplete information.
- Summary:** The Archive of Formal Proofs (AFP) is the central repository for Isabelle formal proofs online. In the first part of the project we re-implemented and redesigned the website. In this project we build on this, adding social features to increase user engagement.
- Background:** Isabelle is a language for writing and validating formal mathematical proofs. The AFP archives Isabelle theorems for use by others. It is a static website and can currently be thought of as read only. All public communication happens externally on the mailing list, Stack Exchange and Zulip. Specific questions are normally emailed directly to the authors making the information inaccessible.

On the right side, a diagram titled "Web server structure" shows "User Requests" pointing to "APACHE HTTP SERVER PROJECT", which then points to "node" (with "Isso a commenting server" and "HUGO" also pointing to it). Below this is the text "Web server structure".

At the bottom right, the poster is attributed to Carlin MacKenzie, supervised by Jacques Fleuriot, and co-supervised by James Vaughan.

Figure E.1: Honours Project Day poster