Technisches

Technologien, Hosting, Webseite, Messages, Bug-Reports, Aufbau auf Vorarbeit

- Link collection for the technical prep work
- Link collection code documentation
- Working with an ethical code assistant - interview with Mark Padgham (Urban Analyst)
- How to Find a GPU Hosting Service – a Guide by Viraaj Akuthota
- Tipps zu KI und LLMs
- Der lange Weg zu einem Local-First-Ökosystem

□□ Link collection for the technical prep work

- Open Source Guide: https://opensource.guide/de/
- Trends auf GitHub: https://github.com/trending
- AlternativeTo, Crowdsourced Software Recommendations: https://alternativeto.net/
- CHAOSS: https://chaoss.community/software/
- ~~You can apply for virtual private servers financed by the Open Tech Fund: https://www.eclips.is/~~ - seems to be currently inactive.

Markdown & co

- *Tables Generator* for Markdown: https://www.tablesgenerator.com/markdown_tables
- *Pandoc* converts various formats (like markdown, HTML, LaTex, docx,...) into each other. Great to build simple tool chains around documents: https://pandoc.org/

□□ Link collection code documentation

To read, listen & watch

- *How to Write Good Documentation*: https://guides.lib.berkeley.edu/how-to-write-good-documentation
- *How to write effective documentation for your open source project*: https://opensource.com/article/20/3/documentation
- *Software Documentation Best Practices [With Examples]*: https://helpjuice.com/blog/software-documentation
- *Ship it!* by Jared Richardson and Will Gwaltney, Jr: https://pragprog.com/titles/prj/ship-it/
- *The Pragmatic Programmer* (Thomas/Hunt, 2020) is a good general introduction to professional software development: Keeping code readable and easy-to-change, learning requirements, utilizing unit- and property tests, basics of securing applications: https://pragprog.com/titles/tpp20/the-pragmatic-programmer-20th-anniversary-edition/
  - There is also a German translation (*Der pragmatische Programmierer*): https://www.hanser-fachbuch.de/fachbuch/artikel/9783446463844
- *Weniger schlecht Programmieren* (Passig/Jander) is similar to "Pragmatic Programmer", but focuses more on code and less on projects: https://dpunkt.de/produkt/weniger-schlecht-programmieren/
- *Command Line Interface Guidelines* are an open-source guide to help you write better command-line programs: https://clig.dev/

Methods & tools

- *Tracer Bullet Development* is a method to get a working version of your software quickly. This allows to evaluate code architecture and user interface early. The method is described on this website: https://growsmethod.com/practices/TracerBullets.html
- *Refactoring Guru* has a lot of resources on common ways (patterns) of how to approach particular problems and how to improve existing code to make it easier to read and to change (refactoring): https://refactoring.guru/
- *12 Factor app* is less about code than about running the code in a way that it can be easily and safely configured, ran and monitored. When you create your project based on modern frameworks (i.e. Laravel) it will often follow such principles: https://12factor.net/

□□ Working with an ethical code assistant - interview with Mark Padgham (Urban Analyst)

Code assistants can be incredibly helpful and save developers a lot of time, but it can take a lot of patience to find the one that works for you. Mark Padgham, our former grantee, was looking for a code assistant that was not only useful but also ethical. Here he explains what he was looking for exactly - and why - and what he ended up using.

What project are/were you working on?

https://urbananalyst.city, based on all software components in https://github.com/UrbanAnalyst

Why were looking for a code assistant?

I needed to code complex routines in rust and compile to web assembly (WASM) to integrate within a Typescript front-end. My usual computer languages are C++ and R, not rust or Javascript. I knew a code assistant could (1) help translate my code ideas from C++ to rust syntax, and (2) help with the WASM integration into Javascript, with which I had no experience at that stage. I wanted a code assistant for the first bit because I thought translating ideas would be faster than me writing everything directly in rust. I needed a code assistant for the second bit, because I really had no idea how to do it.

Are there code assistants that are not based on AI?

I only note that I don't use the term "AI". In this context, there is a distinction between large language models (LLMs), and the equivalent trained on code, which are commonly referred to as "large code models." Simply referring to all as "AI" erases these kinds of important distinctions. (And there are many other, much more important reasons to avoid using that term.) LLMs are also very good at code, and I know many people who use ChatGPT as a code assistant. The model I ended up using is built on CodeLLama (https://huggingface.co/codellama). But it's probably reasonable to suspect that there are no code assistants in the form we know know them that are not based on LLMs or LCMs.

Why was it important to you to use ethical AI? What are the benefits of en ethical AI code assistant?

I have never used GitHub Copilot, because it was obvious to me and many other people from the very beginning that that was trained by scraping the entire contents of GitHub without any consideration of, or respect for, the open source license conditions of code used in that training. https://githubcopilotlitigation.com/ traces the history of the main legal case against Copilot, which is still ongoing. This behaviour is unethical, and I refuse to use any tools trained in such ways.

More general arguments against this kind of wholesale "theft" (generally expressed in legal contexts as "unlawful use") of material for training LLMs have emerged in the case brought by the New York Times against OpenAI: https://theconversation.com/how-a-new-york-times-copyright-lawsuit-against-openai-could-potentially-transform-how-ai-and-copyright-work-221059

How did you conduct your research?

Primarily through reading "model cards" on huggingface.com, and scouring articles they referred to for details of training data. The original Code Llama model is described in the paper linked from https://ai.meta.com/research/publications/code-llama-open-foundation-models-for-code/, with the model card for the current version at https://huggingface.co/codellama/CodeLlama-70b-hf. I followed all links to understand exactly how the training data were obtained, and was satisfied that Code Llama was about as ethically acceptable as you get these days.

Which tool did you end up using?

phind.com. At the time I started using it, it was based on CodeLlama-34B, described in the Phind blog at https://www.phind.com/blog/code-llama-beats-gpt4, with the CodeLlama model card at https://huggingface.co/Phind/Phind-CodeLlama-34B-v2. The Phind model uses additional training on "an internal Phind dataset" of 1.5B tokens. There used to be a mention elsewhere that suggested that they purchased that dataset commercially. The ethics of such datasets and their procurement can of course never be ascertained, but both the initial development of CodeLlama, and the subsequent development of Phind, certainly seem to have been done with far greater ethical awareness than anything displayed by OpenAI or Microsoft. And the majority of training data both for CodeLlama and Phind remain datasets that have been obtained in full respect of open source licensing conditions.

Do you have any other recommendations for developers who are looking for good tools to help them do their work?

Read the enormous number of articles that have been written on the motivations of the major companies who develop LLMs to understand their very specific agendas. And then act against them by refusing to use their tools in favour of alternatives that respect open source licensing conditions. The current state of code assistants, LLMs, and associated hype is insightfully analysed in

https://blog.pragmaticengineer.com/the-ai-developer/. To quote that article, the hype surrounding this technology is, "a well-choreographed show."

How to Find a GPU Hosting Service – a Guide by Viraaj Akuthota

For his project "Human Rights Predictor" (Round 15) our grantee Viraaj Akuthota was looking for a GPU hosting service. Here he explains how he went about it:

To fine-tune models and create embeddings on large corpuses of qualitative data, a high amount of GPU RAM (VRAM) is required. For example, fine-tuning BERT on a dataset of 15k cases that vary in size creates roughly 100k-200k sequences at a 512 token limit. This requires approximately 140 GB of VRAM. This hardware requirement means such tasks cannot be conducted on most consumer-grade machines. I conducted an exercise to hopefully identify an affordable and relatively easy-to-use cloud compute option. During this search, I faced many difficulties. The benefits and disadvantages of the majority of service providers I reviewed can be found in the table below.

Overall, the production system I landed on is to utilize:
· PaperSpace's Core using a Windows Server instance to avoid using the terminal as much as possible.
· Always available Multi-GPU instances, for example, 4 x A6000 Nvidia GPUs with 192 GB VRAM total for roughly $7 USD an hour.
· Approximately $3 USD per month for 50 GB persistent storage, making offline costs negligible.
· For Linux users, they have a Python ML template which will save time installing python, packages, cuda, etc.

Before production, I utilise either Google Colab or HuggingFace:
· For testing fine-tuning or creating embeddings, I believe Google Colab's free T4 instance provides the highest amount of VRAM for any free tier.
· For testing LLMs, HuggingFace's serverless inference free tier allows you to utilize a variety of LLMs such as LLAMA 405B. However, the Pro tier at $9 USD per month increases the rate limit on this inference. I receive approximately 300 API calls per hour.

Provider	Benefits	Disadvantages	GPU Limit

Amazon EC2	• Relatively affordable compared to other cloud providers	• Requires familiarity with AWS • Application for quotas is not straightforward and the approval process takes time	• Essentially unlimited
Amazon Notebooks	• Easy to set up an ML system • Relatively affordable compared to other cloud providers	• Notebooks are limited to certain GPU sizes, essentially under 100GB VRAM. • Even if you have a quota for the underlying resource it will not work for a notebooks	• Under 100GB VRAM
Microsoft Azure		• The registration system and console is sufficiently complicated that I did not utilise this service. • Quota application process did not seem straight forward.	• Essentially unlimited
Google Cloud		• Unable to secure access to a high-end GPU as they were **ALWAYS** unavailable	

Google Colab	• Very easy to use and set up	• Relatively more expensive • Not guaranteed access to the most powerful GPUs that is claimed to be accessible even with premium services	• A100 GPU with 40GB VRAM, if available, which is rare
Paperspace Notebooks	• Very easy to use and set up • Multiple ‘free’ GPU availability with unlimited hours at the premium option		• PaperSpace has plans which provide various systems at 6 hours of continuous use at a mix of free or paid options. The free options still require a base payment plan to be purchased • For the premium plan, a single P5000 15gb VRAM machine is available for free. • A 'core' machine can also be purchased where you can pay per hour without having to pay for a monthly plan. I currently have 4 x A6000 48gb VRAM for $7.56 an hour.
Paperspace Server/Console	• Always available multi-GPU instance • ML template server instances • Easy server setup	• More expensive than the big players • Some of the ML template server instances come with certain issues with libraries	• Essentially unlimited

□□ Tipps zu KI und LLMs

Hier veröffentlichen wir fortlaufend Tipps und Tricks rund um die Entwicklung von KI-Anwendungen:

Zum Training und Finetuning von LLMs gibt es den Unsloth's Instruct Modell Trainer, der kostenlos ist, auf einer ebenfalls kostenlosen Google Colab Instanz betrieben werden kann und sehr gute Ergebnisse produziert. Hier geht es zum Repository:

https://github.com/unslothai/unsloth?tab=readme-ov-file.

□□ Der lange Weg zu einem Local-First-Ökosystem

Software zu entwickeln, die sowohl offline als auch online funktioniert – und das ganz ohne Server –, sollte doch eigentlich nicht allzu schwer sein, oder? Ganz so einfach ist es nicht. Das Team des Projekts Reflection aus Jahrgang 01 gibt in diesem Gastbeitrag einen kurzen Überblick über den aktuellen Stand des Local-First-Ökosystems und einige Probleme, über die sie auf dem Weg gestolpert sind.

Auch wenn Local-First nach einem simplen und sinnvollen Prinzip klingt, steckt der Bereich noch in den Kinderschuhen und es gibt derzeit noch nicht viele Local-First-Apps, die man tatsächlich herunterladen und ausprobieren kann. Selbst für die meisten Entwickler*innen ist unklar, welche Hürden noch überwunden werden müssen, um zu einem ausgereiften Local-First-Tech-Stack zu gelangen.

Nach der Entwicklung und Veröffentlichung einer Beta-Version von Reflection, einem Local-First Texteditor, wissen wir nun genauer, welche Arbeit noch vor uns liegt, um Local-First zur Reife zu bringen. In diesem Beitrag möchten wir einen Überblick darüber geben, wo dieser Stack derzeit steht: Was schon funktioniert, was noch zu tun ist, was Entwickler erwarten können und wo Beiträge am dringendsten benötigt werden.

Niemand sonst ist online

Etwas, das wir zwar erwartet hatten, das in der Praxis aber dennoch nervig ist, ist, dass eine Synchronisierung in einem reinen Peer-to-Peer-Netzwerk nur möglich ist, wenn mindestens ein anderer Peer online ist. In einer kleinen Gruppe bedeutet dies, dass Synchronisierungsmöglichkeiten oft aufeinander abgestimmt werden müssen. Für Notizen bei Treffen ist das meist kein Problem weil alle gleichzeitig online sind, Aber selbst in diesem Fall bedeutet es, dass man die Tagesordnung nicht vorab in ein Pad einfügen und davon ausgehen kann, dass andere sie auch sehen können. Man kann auch nicht davon ausgehen, dass ein*e Kolleg*in, die*der das Treffen verpasst hat, die Notizen später lesen kann. Dieses Problem wird als Nebenwirkung der Datenschutzgarantien von p2panda noch verschärft: Jedes Reflection-Pad kann nur mit anderen Peers synchronisiert werden, die das geheime Passwort kennen (d. h. den Einladungscode in Reflection).

Einige unserer Community-Mitglieder haben schnell Workarounds für dieses Problem entwickelt, beispielsweise indem sie einen Computer ständig online lassen und das Dokument geöffnet halten. Das mag das Problem zwar lösen, ist aber mühsam und offensichtlich keine skalierbare Lösung.

Die langfristige Lösung der meisten Local-First-Projekte für dieses Problem sind sogenannte Support-Nodes. Die Idee dahinter ist, dass es einige Nodes gibt, die (fast) immer online sind und Änderungen von Personen abrufen können, die nur sporadisch online sind. Sie können diese dann an andere weiterleiten, die später online gehen. Diese Support-Nodes können auf einem Server in einem Rechenzentrum laufen, aber auch auf einem Raspberry Pi oder einem ungenutzten Laptop im Wohnzimmer.

Die Implementierung von Support-Nodes ist nicht trivial: Sie erfordert Ende-zu-Ende-Verschlüsselung (damit Support-Nodes, die auf einem Server laufen, die Daten und im Idealfall auch die Metadaten nicht lesen können) sowie eine Reihe von UI/UX-Überlegungen, um sicherzustellen, dass sich Support-Nodes innerhalb des App-Kontexts einfach einrichten, konfigurieren und entfernen lassen. Im Idealfall wird daraus eine standardisierte Infrastruktur, die von verschiedenen Local-First-Apps und -Protokollen gemeinsam genutzt werden kann, sodass ein einziger Support-Node für alle Apps verwendet werden könnte.

Bereits existierende Beispiele sind die Chatmail-Server von DeltaChat, die ausschließlich verschlüsselte E-Mails verarbeiten und diese nach 90 Tagen automatisch löschen, Keyhive sowie die Broker von Nextgraph. Der Ansatz von p2panda besteht darin, `p2panda-encryption` zu nutzen, um geheime Verschlüsselungscodes für eine Gruppe von Benutzer*innen zu generieren, mit denen alle Daten, einschließlich aller Metadaten verschlüsselt werden. Diese werden dann in einer einfachen Key-Value-Datenbank auf dem Support-Node gespeichert, von wo aus andere Teilnehmer*innen sie abrufen können.

Als provisorische Kurzzeitlösung erwägen wir, Reflection so zu erweitern, dass es auch in der Kommandozeile läuft, damit man eine Instanz davon auf einem Server ausführen kann. Dies wäre eine einfache Möglichkeit, die Verfügbarkeit sicherzustellen, allerdings mit einem wesentlichen Nachteil: Der Inhalt der Dokumente würde unverschlüsselt auf dem Server gespeichert werden.

Vereinzelte Dokumente

Jedes Reflection-Dokument befindet sich derzeit in einem eigenen Netzwerk aus Peers, was bedeutet, dass Nutzer*innen jedem einzelnen Dokument, das sie synchronisieren und aufrufen möchten, manuell beitreten müssen. Besonders für Projekte, bei denen eine Gruppe von Personen an mehreren Dokumenten zusammenarbeiten muss, ist das sehr umständlich.

Im Reflection-Team haben wir beispielsweise zwei zentrale gemeinsame Dokumente: eines mit unseren Besprechungsnotizen und ein separates für unsere Roadmap. Außerdem erstellen wir regelmäßig Ad-hoc-Dokumente für den Entwurf von Blogbeiträgen, Förderanträgen und ähnlichen Aufgaben. Da all diese Dokumente organisatorisch zusammengehören, ist es sinnvoll, sie stets zusammen zu synchronisieren. Derzeit gibt es jedoch keine Möglichkeit, in all diesen Dokumenten gleichzeitig „online“ zu sein, es sei denn, man öffnet mehrere Fenster der Reflection-App.

Wir möchten zwar den Aufwand für die Synchronisierung von Gruppen gemeinsamer Dokumente verringern, wollen aber nicht jedes einzelne Dokument, an dem wir jemals gearbeitet haben,

ständig synchronisieren. Angesichts des Peer-to-Peer-Charakters der App ist es wichtig, Netzwerkverbindungen nur gezielt herzustellen, um beispielsweise eine versehentliche Offenlegung der IP-Adressen der Teilnehmer*innen zu vermeiden.

Da p2panda sehr flexibel darin ist, wie die Daten zueinander in Relation stehen, wären „Spaces“, die mehrere Dokumente in einem Kontext gruppieren, bereits möglich. Indem mehrere „Sync-Handles“ am Laufen gehalten werden, würde p2panda auch dann im Hintergrund weiter synchronisieren, wenn die Dokumente von der*die Benutzer*in geschlossen sind. Aus Backend-Sicht wäre es somit bereits recht einfach, einen „Shared Folder“-Workflow zu ermöglichen, würde aber auch einiges an Arbeit an der Benutzeroberfläche der App erfordern.

In Kombination mit p2pandas Gruppenverwaltungs- und Verschlüsselungslösung (p2panda-spaces) könnten wir für jeden „Space“ angemessene Zugriffskontrollen einrichten, Mitglieder hinzufügen und entfernen sowie alle Dokumente ausschließlich für diese Mitglieder verschlüsseln. Dies befindet sich bei p2panda derzeit noch im Versuchsstadium. Daher möchten wir abwarten, bis sich die APIs stabilisiert haben, bevor wir sie in Reflection integrieren.

Letztendlich wäre es schön, dies als Teil eines Systemdienstes oder eines Hintergrund-Daemons zu realisieren, damit wir Dokumente auch dann synchronisieren können, wenn die Reflection-App geschlossen ist. All das würde jedoch eine sorgfältige Risikoanalyse voraussetzen.

Geräte entfernen

Wir haben kürzlich eine Reihe von „Developer Labs“ Treffen für Entwickler*innen von GTK-Apps veranstaltet. Ein Beispiel, das dabei zur Sprache kam, war die Synchronisierung der Liste der Feed-Abonnements in einem RSS-Reader zwischen mehreren Computern. Aus Sicht des Datenmodells ist dies einfach und benötigt kein komplexes „Conflict-free replicated data type“ (CRDT), aber man braucht eine Möglichkeit, alte Geräte zu entfernen (z. B. wenn man sie verliert).

Mit dem aktuellen Zugriffsmodell, das wir in Reflection verwenden, hat man, sobald man den Einladungscode kennt, vollen Zugriff. Dieser kann von anderen Geräten nicht widerrufen werden. Zwar gibt es Workarounds für dieses Problem, die potenziell auf App-Ebene implementiert werden könnten (z. B. das Dokument ohne das Gerät, das man entfernen möchte, neu zu erstellen), aber eine saubere Lösung erfordert ein ausgefeilteres Zugriffskontrollsystem.

p2panda ist dabei, Lösungen für eine fein abgestufte Zugriffskontrolle einzuführen, die sowohl die direkte Peer-to-Peer-Synchronisation als auch vollständig verschlüsselte Support-Knoten berücksichtigen. Wir hatten ursprünglich geplant, an einer Testintegration in Reflection zu arbeiten, haben diese Arbeit jedoch aus Zeitgründen verschoben. Die Stabilisierungsarbeiten an diesen APIs laufen derzeit noch. Wenn dies also in einer App benötigt wird, empfehlen wir, abzuwarten, bis dieser Prozess abgeschlossen ist.

Ein anderer Ansatz kann darin bestehen, Berechtigungen mit einem Ablaufdatum zu versehen. Ein Beispiel hierfür ist Meadowcap von Willow. Meadowcap unterstützt keine manuelle Widerrufung,

sondern erlaubt nur die Gewährung von Zugriff für ein bestimmtes Zeitfenster, das jedoch verlängert werden kann. UCAN ist eine ausgefeiltere Version eines Berechtigungssystems und verfügt über eine Widerrufungsfunktion (unter Verwendung von Ketten). All diese Ansätze könnten mit dem p2panda-Stack kombiniert werden.

Lange Join-Codes

Um derzeit in Reflection zusammenzuarbeiten, teilt man einen 64 Zeichen langen alphanumerischen “Beitrittscode“. Dieser entspricht unserer „URL“ zu einem Dokument und dient gleichzeitig als Passwort. 64 Zeichen sind zu lang, um sie laut vorzulesen, wenn man persönlich zusammen sitzt oder telefoniert. Das bedeutet, dass man einen Gruppenchat oder einen ähnlichen, anderen Kommunikationsweg benötigt, um ihn zu teilen, was umständlich sein kann.

Eine mögliche Lösung ist, den Beitrittscode über einen QR-Code zu teilen. Das funktioniert gut auf Mobilgeräten, ist aber für den Desktop keine Option.

Eine andere Möglichkeit ist, den Beitritt über kürzere Codes zu ermöglichen, z. B. nur 4 alphanumerische Zeichen wie bei Toolkitty. Dies ist weniger sicher, da diese Codes viel leichter durch Brute-Force-Angriffe erraten werden können. In Kombination mit Zugriffskontrollen oder in Situationen, in denen Dokumente nur in einem lokalen Netzwerk geteilt werden, kann das jedoch akzeptabel sein. Es besteht aber auch die Gefahr von Unstimmigkeiten, wenn zwei Dokumente denselben kurzen Beitrittscode verwenden; dies müsste daher in der Benutzeroberfläche irgendwie berücksichtigt werden.

Privatsphäre der IP-Adresse

Ein grundlegendes Problem bei Peer-to-Peer-Netzwerken besteht darin, dass man sich direkt mit vielen verschiedenen Endgeräten verbindet und nicht mit einem einzigen Server. Das bedeutet, dass alle Teilnehmer*innen des Netzwerks die IP-Adressen der jeweils anderen kennen. Bei der Zusammenarbeit mit einer kleinen Gruppe vertrauenswürdiger Partner*innen an einem Projekt ohne besondere Sicherheitsanforderungen ist das normalerweise kein Problem. Je nach Anwendungsfall kann das Durchsickern von IP-Adressen jedoch gefährlich sein, sodass es keine gute Idee ist, Peer-to-Peer-Apps ohne zusätzliche Sicherheitsmaßnahmen zu nutzen.

Das ist ein wichtiger Grund warum wir momentan nicht empfehlen, Reflection in Situationen zu verwenden, in denen der Einladungscode öffentlich geteilt wird, z. B. in sozialen Medien. Es ist auch der Grund, warum wir es noch nicht für schutzbedürftige Personen, wie z. B. Aktivist*innen, empfehlen würden. IP-Adressen können dazu verwendet werden, Teilnehmer*innen im Netzwerk zu identifizieren und ins Visier zu nehmen (ähnlich wie Anwaltskanzleien, die Personen wegen Torrenting verfolgen), da sie in vielen Fällen zu einem geografischen Standort oder zur Inhaber*in der Internetverbindung zurückverfolgt werden können.

Die naheliegendste Abhilfe für dieses Problem ist die Verwendung einer Anonymisierungstechnologie wie Tor, I2P, Katzenpost oder Veilid. Wenn der gesamte Datenverkehr anonymisiert ist, ist das Problem mit der IP-Adresse weitgehend hinfällig. Glücklicherweise hat unsere Netzwerkbibliothek iroh kürzlich Unterstützung dafür hinzugefügt, sodass dieses Problem in Zukunft wahrscheinlich weniger relevant sein wird.

Privat arbeiten

Echtzeit-Zusammenarbeit ist zwar großartig, aber manchmal möchte man sich vom „Fishbowl-Effekt“ (dem Wissen, dass andere beobachten, was man tut) abkoppeln und eine Weile alleine arbeiten.

In Reflection ist das bereits heute möglich, indem man die App in den Offline-Modus schaltet. Wenn man jedoch wieder online geht, hat man keine Kontrolle darüber, wie die eigenen Änderungen mit denen der anderen zusammengeführt werden. Das ist nicht immer ein Problem, aber besonders wenn man für längere Zeit offline ist, herrscht Ungewissheit darüber, was passieren wird, wenn man wieder online geht. Was, wenn meine Änderungen keinen Sinn mehr ergeben? Was, wenn die anderen das Dokument komplett umgeschrieben haben, während ich weg war?

Das kam auch in unseren Anwender*innenbefragungen zur Sprache, in denen die Teilnehmer*innen große Unsicherheit hinsichtlich der Zusammenführung nach der Rückkehr in den Online-Modus äußerten. Einige dachten, dass alle ihre Änderungen gelöscht würden, andere vermuteten, dass ihnen die Unterschiede angezeigt würden, aber fast niemand erwartete das tatsächliche Verhalten (nämlich die automatische Zusammenführung mit einem CRDT).

Um diesen Unsicherheiten entgegenzuwirken, halten wir es für sinnvoller, eine explizite Möglichkeitanzubieten, um eine eigene Kopie des Dokuments abzuzweigen und privat daran zu arbeiten. Dies würde einen klaren „Zusammenführungs“-Vorgang ermöglichen, wenn man mit seinem privaten Entwurf fertig ist und zur Zusammenarbeit in Echtzeit zurückkehren möchte. Es gibt Forschungen in dieser Richtung, wie beispielsweise Ink&Switchs Upwelling, aber bisherkeine fertigen Implementierungen, die produktiv eingesetzt werden.

Wer ist „Purple Horse“?

Derzeit ist es in Reflection nicht möglich, die Gerätenamen zu ändern, die automatisch jedem Gerät zugewiesen werden („Purple Horse“, „Green Panda“ usw.). Das hat sich in der Praxis als Problem erwiesen, da es den Nutzer*innen schwerfällt, sich zu merken, welches Tier zu wem gehört.

Wir könnten zwar die Möglichkeit einführen, dass jeder einfach seinen eigenen Namen und Benutzer-Avatar ändern kann. Das würde aber gewisse Risiken mit sich bringen, die wir nicht eingehen wollen, solange wir keinen langfristigen Plan haben, wie wir mit Identitäten umgehen.

Das große Problem bei benutzer*innendefinierten Identitäten, insbesondere in einem Peer-to-Peer-Kontext ohne zentrale Instanz, die Benutzer*innen verifizieren kann, ist Identitätsbetrug. Es ist vergleichsweise einfach, den Namen und den Avatar einer anderen Person zu kopieren, diese im eigenen Konto zu nutzen und sich als diese Person gegenüber derenKontakten auszugeben.

Die Alternative zu benutzer*innendefinierten Identitäten besteht darin, dass alle ihr eigenes, persönliches Adressbuch mit den Namen all ihrer Kontakte führen. So kann Identitätsdiebstahl verhindert werden, da der sichtbare Name der Person nicht von einem potenziellen Angreifer festgelegt werden kann. Dies wird als „Pet-Name-System“ bezeichnet. Beispiele hierfür sind Backchannel und Spritely. Der Nachteil dieses Modells ist, dass es für alle Beteiligten sehr aufwendig ist, Adressbücher manuell anzulegen und zu verwalten.

Derzeit gibt es keine umfassende und für alle Fälle gangbare Lösung für dieses Problem, insbesondere bei großen Gruppen. Eine mögliche kurzfristige Lösung für Reflection wäre das Hinzufügen eines Adressbuchs, in dem Nutzer*innen optional Spitznamen für ihre Kontakte registrieren können. Es braucht aber weitere Forschung, um eine Benutzer*innenerfahrung zu entwickeln, die sowohl sicher als auch komfortabel ist.

Um diesen Onboarding-Prozess zu vereinfachen, wäre ein einheitliches Adressbuch hilfreich, das von mehreren Apps gemeinsam genutzt wird. Wir führen derzeit Gespräche darüber, wie eine System-API unter Linux hierfür aussehen könnte. DeltaChat ist ein gutes Beispiel für eine solche „Host-Umgebung“ mit einem einheitlichen Adressbuch (deine Chat-Kontakte), in der Gruppen erstellt werden können, um WebXDC-Apps auszuführen.

Und nun?

Im Gespräch mit App-Entwickler*innen haben wir festgestellt, dass es für viele, die neu im Bereich „Local-First“ sind, schwierig sein kann, sich einen Überblick darüber zu verschaffen, was tatsächlich noch fehlt, um eine Local-First-App zu entwickeln. Es sind viele verschiedene Technologien beteiligt, von denen einige recht ausgereift sind (z. B. CRDTs), andere jedoch noch experimentell (z. B. Zugriffskontrollen oder Identitätsmanagement).

Wir hoffen, dass dieser Beitrag dabei helfen kann, zu zeigen, was heute bereits umgesetzt werden kann, und vielleicht sogar einige Leute dazu zu begeistern, an den noch fehlenden Aspekten zu arbeiten.

Wenn du Interesse hast, mitzumachen, schreib uns gerne eine Nachricht oder folge Modal (@modal@mastodon.design) und p2panda (@p2panda@autonomous.zone) auf Mastodon.