Richtlinie zum „Software-Sprint“

RoaringRoster – kollaborative Dienst- und Abwesenheitsplanung

Schlussbericht

Zuwendungsempfänger:

Michael Kamphausen

Das diesem Bericht zugrundeliegende Vorhaben wurde mit Mitteln des Bundesministeriums für Bildung und Forschung unter dem Förderkennzeichen 01IS23S40 gefördert. Die Verantwortung für den Inhalt dieser Veröffentlichung liegt beim Autor.

Kurze Darstellung der Aufgabenstellung und Motivation

Die Verarbeitung sensibler Daten in der Cloud birgt erhebliche Sicherheitsrisiken, sei es durch Angriffe krimineller Akteure, Einsehbarkeit für technische Dienstleister, technische Ausfälle, Einstellung des Cloud-Dienstes, Verkauf oder Abwicklung des Anbieters oder Vendor-Lock-In. Dennoch wollen wir als Anwender:innen auf bequeme Kollaborationsfunktionen beim gemeinsamen Arbeiten mit Daten nicht verzichten. Daher plante ich mit diesem Projekt eine kollaborative Local-First-App zu entwickeln, die Daten lokal, dezentral und verschlüsselt speichert, uneingeschränkt offline funktioniert und die Daten mit App-Instanzen auf anderen Geräten bequem Ende-zu-Ende-verschlüsselt synchronisiert. Das sorgt für optimale Resilienz und Datenschutz und vermeidet Single-Points-of-Failure.

Gleichzeitig arbeiten und entscheiden immer mehr Teams und Organisationen in Arbeitswelt und Zivilgesellschaft selbstorganisiert. Leider forcieren digitale Personalplanungs-Tools oftmals noch Kontrolle, Überwachung, hierarchische Entscheidungen und intransparentes Herrschaftswissen. Deshalb wurde im Rahmen des Projekts eine kollaborative Dienstplanungs-App entwickelt, mit der Teams ihre Arbeitszeiten und Abwesenheiten kollegial, transparent und kommunikativ planen können. Das Tool ermöglicht eine bequeme Kollaboration, bei der die Daten zwischen den Teilnehmenden sicher synchronisiert werden.

Die App soll plattformübergreifend nutzbar sein, daher plante ich sie als Electron-App für Desktop-Betriebssysteme und als Cordova-App für iOS und Android umzusetzen. Die Benutzeroberfläche plante ich mit webbasierten Technologien in TypeScript mit den Open-Source-Frameworks Vue.js und Quasar zu entwickeln, einschließlich der Unterstützung von Mehrsprachigkeit. Als lokale

Datenbank wollte ich IndexedDB bzw. SQLite einsetzen, wobei die Open-Source-Bibliotheken Dexie und Dexie-encrypted die Handhabung und Verschlüsselung der Datenbank ermöglichen sollten. Ich beabsichtigte Offline-Fähigkeit zu gewährleisten, indem die lokale Datenbank eine vollständige Kopie sämtlicher Daten enthält, auf die die Benutzer:in Zugriff hat.

Die App sollte die Nachvollziehbarkeit der Änderungshistorie samt Bearbeiter:in ebenso ermöglichen wie das Erkennen und Zusammenführen abweichender Änderungen (Versionskonflikte) ohne zentrale Instanz. Daher plante ich CRDT (Conflict-free replicated data types) als Datenstruktur und die Open-Source-Bibliothek Automerge einzusetzen, die für ein solches verteiltes System ideal geeignet sind.

Auch die Benutzer:innen-Authentifizierung und Zugriffsrechte wollte ich bevorzugt dezentral in der App verwalten mit einer Signature-Chain und der Open-Source-Bibliothek Auth. Diese dezentrale Benutzer:innenverwaltung soll eine direkte Datenübertragung zwischen den verschiedenen Geräten des Teams ermöglichen, so dass ein Server nicht mehr zwingend benötigt wird. Optional kann aber ein Server zum Einsatz kommen, etwa als Zwischenspeicher bei der Übertragung, da zu erwarten ist, dass oft nur ein Teil der Teammitglieder gleichzeitig mit der App online ist.

Am Beispiel dieser Dienstplanungs-App wollte ich evaluieren, inwiefern diese noch sehr jungen Local-First-Technologien im Alltag produktiv eingesetzt werden können, und in welchen Bereichen möglicherweise Einschränkungen für die Funktionalität bestehen können.

Die wichtigsten Meilensteine des Projekts waren:

- **App-Setup**: Einrichtung der Cordova- und Electron-App samt Build-Prozess und Auto-Update-Funktion
- **Offline-Fähigkeit**: Persistenz der verarbeiteten Daten in lokaler, verschlüsselter Datenbank
- **Auth**: Authentifizierung, Verwaltung der Zugriffsrechte und Einladung neuer Teammitglieder
- **Datenübermittlung**: Ende-zu-Ende-verschlüsselter Datenabgleich zwischen den Endgeräten
- **Änderungshistorie** und Versionskonflikte darstellen
- **Mitarbeiter:innen-Stammdaten** und Arbeitszeitkonto erfassen und bearbeiten
- **Abwesenheiten**: Urlaub, Krankmeldungen und Fortbildungen einreichen und übersichtlich visualisieren
- **Dienstplanung**: Schichten, verfügbare Arbeitskräfte und Arbeitszeiten kollaborativ planen
- **Veröffentlichung**: Projektwebseite erstellen, App-Distribution als Download via Webseite und App Stores, Vorbereitung Demo-Day

Beitrag des Projektes zu den Zielen der Förderinitative „Software-Sprint“

RoaringRoster unterstützt selbstorganisierte Teams und Organisationen, ihre Arbeitszeiten und Abwesenheiten kollegial und kollaborativ zu planen und abzustimmen, und dabei den arbeitsorganisatorischen Anforderungen und menschlichen Bedürfnissen der Teammitglieder gleichermaßen gerecht zu werden. Jedes Teammitglied ist zugleich Nutzer:in der App und hat somit gleichermaßen Einblick und Teilhabe an allen relevanten Daten und kann an der Planung partizipieren.

Zusätzlich ermöglicht die App den Teams die Kontrolle und Hoheit über die eigenen Daten, da diese immer lokal, sicher und dezentral auf den eigenen Geräten gespeichert bleiben und zwischen diesen ausschließlich Ende-zu-Ende-verschlüsselt übertragen werden zum Schutz der Privatsphäre aller Teammitglieder. Dieser Ansatz ermöglicht außerdem eine zuverlässige Offline-Nutzung der App, sowie Unabhängigkeit vom Anbieter der App, da diese anders als Cloud-Dienste auch dann nutzbar bleibt, wenn der Anbieter seinen Dienst einstellt oder technische Probleme hat.

Schließlich profitieren die Teams und Organisationen davon, dass RoaringRoster eine vollwertige Open-Source-Alternative zu proprietärer Personalplanungs-Software wird, zu der sie selbst neue Funktionen beitragen und Anpassungen an den eigenen Bedarf vornehmen können und darüber hinaus Transparenz über die Funktionsweise, Sicherheit und Vertrauenswürdigkeit der Software erhalten.

Ich erkenne in der kollegialen Arbeitszeitplanung auf Augenhöhe, der Kontrolle über die eigenen Daten und deren lokale Verfügbarkeit, sowie in der Veröffentlichung unter einer freien Open-Source-Lizenz zur Transparenz und Kooperation eine Ermächtigung der App-Benutzer:innen im Sinne von „Civic Tech", die der Zielsetzung des „Software-Sprints" entspricht.

Ausführliche Darstellung der Ergebnisse

Am Anfang der Projektarbeit stand die Einrichtung und Konfiguration der App für die Betriebssysteme iOS und Android mit Hilfe des Cordova-Frameworks, sowie für Windows und macOS mit dem Electron-Framework. Eine Installation unter Linux wäre mit geringem Mehraufwand möglich, konnte aber im Projektzeitraum nicht getestet werden. Die Open-Source-Frameworks Electron und Cordova ermöglichen es, auf Webtechnologien basierende Apps wie RoaringRoster als installierbare App statt im Browser zu nutzen und bieten Schnittstellen zu Funktionen der Betriebssysteme. Ich habe die Benutzeroberfläche basierend auf Webtechnologien und den Open-Source-Frameworks Vue.js und Quasar eingerichtet und Unterstützung für Mehrsprachigkeit auf Englisch und Deutsch hinzugefügt. Darüber hinaus habe ich das Layout und die Navigation der App konzipiert und umgesetzt.

Um den rechtlichen Anforderungen an eine App gerecht zu werden, habe ich eine Datenschutzerklärung und ein Impressum erstellt, sowie die Lizenztexte der App und aller verwendeten und eingebundenen Open-Source-Bibliotheken einschließlich Namensnennung der Urheber:innen in die App integriert. Ich habe für RoaringRoster die AGPL-Lizenz gewählt und ein Contributor License Agreement (CLA) für potenzielle Beitragende zu diesem Open-Source-Projekt erstellt, um auf nicht vorhersehbare, zukünftige Anforderungen an die Lizenz reagieren zu können, sowie einen englischsprachigen Leitfaden für Beitragende verfasst. Ich habe bei Github eine neue Organisation für RoaringRoster und Repositories für die App und die CLA angelegt.

Ich habe Build-Skripte geschrieben, die für jedes unterstützte Betriebssystem eine passende App-Version auf Knopfdruck automatisiert bauen, kompilieren, signieren und auf die Projekt-Webseite und zu Apples App-Store-Connect-Dienst hochladen, wo sie jeweils von Tester:innen und Anwender:innen heruntergeladen werden kann. In die RoaringRoster-App habe ich eine Benachrichtigungs-Funktion integriert, die die Benutzer:in automatisch informiert, sobald ein App-Update für ihr System zur Verfügung steht und mit der er eine Selbst-Aktualisierung der App veranlassen kann. Somit konnte ich den Meilenstein „App-Setup" abschließen.

Ich habe eine Corporate Identity für RoaringRoster ausgearbeitet einschließlich App-Logos, App-Launch-Screens und Farbschema. Ich habe eine Projekt-Webseite erstellt und hierfür Texte formuliert, Illustrationen ausgewählt, sowie eine Newsletter-Registrierung, Domain, E-Mail-Adressen und DNS eingerichtet. Für den Demo-Day habe ich Plakate und Visitenkarten entworfen und vorbereitet. Damit konnte ich wesentliche Teilaufgaben des Meilensteins „Veröffentlichung" umsetzen, mit Ausnahme von User-Testing und einer Veröffentlichung in den App-Stores aus zeitlichen Gründen. Stattdessen wird nach dem Förderzeitraum zunächst eine geschlossene Beta-Testphase beginnen.

Ich habe die persistente Speicherung der in der App verarbeiteten Daten in einer lokalen IndexedDB- oder SQLite-Datenbank entwickelt. Diese Datenbank enthält eine vollständige Kopie sämtlicher Daten, auf die die Benutzer:in Zugriff hat, um die Offline-Fähigkeit zu gewährleisten. Die Daten werden verschlüsselt gespeichert mit dem Hashwert des Passworts, mit dem sich die Benutzer:in lokal in der App anmeldet, und welches somit zur Entschlüsselung zwingend benötigt wird. Ich konnte den Meilenstein „Offline-Fähigkeit" vollständig umsetzen.

Das Datenmodell basiert auf CRDT (Conflict-free replicated data types) als Datenstruktur, bei der die gesamte Änderungshistorie eines Datenobjekts bzw. Dokuments erhalten bleibt. Damit lässt sich jede Änderung rückwirkend nachvollziehen, während gleichzeitig das deterministische Zusammenführen abweichender Änderungen der Daten (Versionskonflikte) ohne zentrale Instanz ermöglicht wird. Zur Verwaltung der CRDT-Daten habe ich die Open-Source-Bibliothek Automerge in die App integriert, sowie eine Komponente für die Benutzeroberfläche entwickelt, in der die Änderungshistorie des aktuell geöffneten Dokuments visualisiert wird und ablesbar ist, welche Benutzer:in wann welches Datenfeld geändert hat. Aus zeitlichen Gründen konnte ich eine automatische Erkennung von Versionskonflikten, die in seltenen Fällen auftreten können, und ihre Darstellung als Warnhinweis in der Benutzeroberfläche nicht abschließend umsetzen. Folglich ist der Meilenstein „Änderungshistorie und Konfliktlösung" nicht vollständig erreicht, wobei ich jedoch die wichtigsten Teilaufgaben abschließen konnte.

Die wichtigste Funktion von RoaringRoster ist die Dienstplanung, mit der App-Benutzer:innen kollaborativ die Arbeitszeiten ihres Teams planen können. Ich habe hierfür die Benutzeroberfläche und das Datenmodell konzipiert und umgesetzt: Grundlage ist eine Wochenansicht, zu der die Nutzer:in die benötigten Arbeitsschichten mit den jeweiligen Arbeitszeiten, der benötigten Mitarbeiter:innen-Anzahl und Qualifikation hinzufügt. Gleichzeitig wird visualisiert, welche Teammitglieder zu welchen Zeiten in der Woche verfügbar sind. Die Anwender:in kann nun sich selbst und verfügbare Kolleg:innen zu jeder offenen Position einer Schicht zuweisen. Darüber hinaus kann ein Wochen-Dienstplan als Vorlage für eine neue Woche übernommen werden, sowie ein durchgeplanter Dienstplan als genehmigt markiert und somit verbindlich werden. Aus Zeitmangel konnte ich die Benutzeroberfläche, in der die Teammitglieder ihre Verfügbarkeit für die Arbeit verwalten können, noch nicht umsetzen, so dass die Benutzeroberfläche der Dienstplanung bisher nur Beispiel-Daten anzeigt. Ebenfalls aus Zeitgründen fehlt noch eine Integration mit den Daten anderer Komponenten der App wie der Abwesenheitsplanung. Somit konnte ich zwar eine Mehrheit, aber nicht alle Teilaufgaben für diesen Meilenstein umsetzen.

Bei der Abwesenheitsplanung können die Teammitglieder für sich selbst und ihre Kolleg:innen Urlaubszeiten, Arbeitsunfähigkeit bzw. Krankmeldung, Fortbildungen und Abwesenheiten aus anderen Gründen in eine Monatsübersicht eintragen, in der die Abwesenheiten aller Teammitglieder

übersichtlich visualisiert werden. Ich habe hierfür die Benutzeroberfläche und das Datenmodell konzipiert und entsprechend der Planung umgesetzt. Damit ist der Meilenstein „Abwesenheiten" erreicht.

Desweiteren habe ich Benutzeroberflächen und Datenmodelle konzeptionell entworfen und umgesetzt, mit denen Mitarbeiter:innen-Stammdaten, bestehend aus Kontaktdaten und vertraglichen Informationen, erfasst und bearbeitet werden können. Während der Konzeption der Komponenten für Stammdaten, Dienst- und Abwesenheitsplanung wurde mir bewusst, dass es aus Nutzer:innen-Sicht einen Bedarf für ein personalisiertes Übersichts-Dashboard gibt. An diesem kann die Anwender:in dringende Neuigkeiten und relevante Informationen für sie und das Team auf einen Blick erkennen, wie beispielsweise eine kurzfristige Krankmeldung, Anfragen zum Schicht-Tausch oder die eigenen Arbeitszeiten der kommenden Woche. Ich habe für dieses Dashboard ein Konzept ausgearbeitet und eine Benutzeroberfläche prototypisch mit Beispiel-Daten umgesetzt. Geplant war für diesen Meilenstein auch die Verwaltung von Arbeitszeitkonten durch die Teammitglieder, was ich nicht umgesetzt habe, sowohl aus Zeitmangel und weil ich den Bedarf für das Dashboard für die Dienst- und Abwesenheitsplanung höher eingestuft habe. Daher habe ich den Aufgabenbereich für den Meilenstein „Mitarbeiter:innen-Stammdaten" erweitert und im Hinblick auf die ursprüngliche Planung überwiegend, aber nicht vollständig erreicht.

Bei meiner Arbeit am Meilenstein „Datenübermittlung" wurde ich darauf aufmerksam, dass von der Open-Source-Community des Automerge-Projekts kurz zuvor mehrere Erweiterungen für das lokale, persistente Speichern und für die Datenübertragung der CRDT-Daten entwickelt wurden, sogenannte Storage- und Network-Adapter. Die Architektur dieses Ansatzes bietet durch einheitliche Schnittstellen für die Adapter eine gute Wiederverwendbarkeit sowie Flexibilität, eigene Adapter für verschiedene Speicher- und Übertragungstechniken zu entwickeln oder einen der vorhandenen Adapter zu nutzen. Das macht es leicht, verschiedene Techniken durch den Austausch des jeweils verwendeten Adapters auszuprobieren, und im Fall der Network-Adapter auch mehrere Techniken parallel zu nutzen. Da es effizienter erscheint, so eine offene Plattform und die Synergien der Community zu nutzen, als für jedes App-Projekt eigenen, nur eingeschränkt wiederverwendbaren Storage- und Network-Programmcode zu schreiben, habe ich den Einsatz dieser Plattform für RoaringRoster evaluiert. Aufgrund des positiven Ergebnisses entschied ich mich dafür, RoaringRoster auf dieser Architektur aufzubauen. Ich habe die Datenübermittlung durch die Integration des Network-Adapters automerge-repo-network-websocket in RoaringRoster und die Installation des zugehörigen WebSocket-Servers automerge-repo-sync-server zwischen mehreren App-Clients erfolgreich testen können. Der Server übernimmt dabei lediglich die Funktion eines Zwischenspeichers, der gewährleistet, dass jeder App-Client Änderungen synchronisieren kann, auch wenn kein anderer Client zu dem Zeitpunkt online ist. Die Adapter-basierte Architektur ermöglicht und vereinfacht dabei für die Zukunft die Integration zusätzlicher Übertragungstechniken in RoaringRoster, etwa eine dezentrale Übertragung durch einen noch zu entwickelnde Bluetooth-Network-Adapter oder einen WebRTC-Network-Adapter, was jedoch im Förderzeitraum zeitlich nicht umsetzbar war. Der Einsatz der Automerge-Adapter-Architektur erforderte, den Programmcode für die Speicherung in einer lokalen IndexedDB-Datenbank zu ersetzen durch den existierenden IndexedDB-Storage-Adapter, was problemlos möglich war, wobei der Adapter noch keine Verschlüsselung der Daten unterstützt. Eine Erweiterung um diese Funktion oder die Entwicklung eines neuen IndexedDB-Storage-Adapters mit Verschlüsselung basierend auf dem Programmcode in RoaringRoster ist möglich, konnte aber zeitlich noch nicht umgesetzt werden. Der Meilenstein „Datenübermittlung" ist zwar nicht vollständig implementiert, da die Datensynchronisierung noch

nicht in alle Komponenten der Benutzeroberfläche integriert ist, aber das technische Konzept dafür habe ich während der Förderung ausgearbeitet, umgesetzt und erfolgreich erprobt.

Ich habe eine Benutzeroberfläche für die Registrierung und den Login lokaler Benutzer:innen in der RoaringRoster-App entwickelt. Damit ermögliche ich zusätzlich Mehrbenutzerfähigkeit auf dem gleichen, gemeinsam genutzten Endgerät. Ich habe die Open-Source-Bibliothek Auth in die App integriert, um die dezentrale Verwaltung der Zugriffsrechte und autorisierter Geräte und Teammitglieder auf der Basis einer Signature-Chain zu ermöglichen. Allerdings habe ich diese Teilaufgabe aus Zeitmangel noch nicht abschließen können, ebenso wie das Einladen und Autorisieren zusätzlicher Teammitglieder und Geräte, sowie die Anwendung einer Ende-zu-Ende-Verschlüsselung auf die übertragenen App-Daten, die ebenfalls in den Anwendungsbereich der Signature-Chain fällt, die hierfür die Public-Key-Infrastruktur bereitstellt. Das technische Konzept hierfür habe ich während der Förderung ausgearbeitet und mit der Umsetzung begonnen, aber letztere noch nicht abgeschlossen und getestet. Für die Datenübertragung mittels WebSockets ist ein erweiterter Websocket-Server zur Unterstützung von Signature-Chains zur Überprüfung der Berechtigung von Datenänderungen erforderlich, für den sich das Open-Source-Projekt auth-syncserver anbietet. Ich habe diesen Server installiert, konfiguriert und angepasst. Somit ist der Meilenstein „Auth" zum Ende der Förderung nur teilweise erreicht, da einige Teilaufgaben umgesetzt und andere noch offen sind.

Zielgruppe, Nutzen und mögliche Weiterentwicklungen

Zur Zielgruppe von RoaringRoster gehören Unternehmen und Organisationen, die sinnstiftende New-Work-Arbeitsformen einführen, bei denen Mitarbeitende rollenbasiert statt in klassischen hierarchischen Strukturen mehr Verantwortung übernehmen und Entscheidungen gemeinsam auf Augenhöhe treffen. Dies sorgt für attraktivere Arbeitsbedingungen und höhere Mitarbeiterzufriedenheit. Desweiteren zählen auch Vereine, Projektgruppen, Genossenschaften, Kollektivbetriebe, sowie Familien zur Zielgruppe, die ohnehin kooperativ und selbstorganisiert gemeinsame Interessen verfolgen.

Das Projekt RoaringRoster ermöglicht es ihnen allen, ihre Arbeitszeiten und Abwesenheiten auf eine kooperative Weise zu planen, die ihren partizipativen Entscheidungs- und Kommunikationsstrukturen entspricht, und dabei alle Informationen und Entscheidungen für alle Teammitglieder transparent und nachvollziehbar macht. Gleichzeitig bietet ihnen dieses Projekt echte Kontrolle, Sicherheit, Privatsphäre und Verfügbarkeit über die eigenen Daten durch die dezentrale Local-First-Architektur, Ende-zu-Ende-Verschlüsselung und Offline-Nutzbarkeit.

Da der Programmcode von RoaringRoster der Open-Source-Community transparent und überprüfbar zur Verfügung steht, kann dies das Vertrauen in die Sicherheit der Software und den Schutz der verarbeiteten Daten stärken. Darüber hinaus ermöglicht es Teams, aus eigener Initiative die App um zusätzlich benötigte Funktionen, Anpassungen oder weitere Datenübertragungstechniken zu erweitern, und damit gleichzeitig zu Synergie-Effekten der Community beizutragen. Ebenso habe ich meinerseits während der Förderung zur Verbesserung des Open-Source-Projekts Cordova beigetragen, als ich beim Testen von RoaringRoster ein Problem in Cordova entdeckte, das die Ausführung von WebAssembly-Programmcode verhinderte, so dass ich eine Lösung entwickelte, die nun Bestandteil des Cordova-Projekts ist.

Ich beabsichtige weiter an RoaringRoster zu arbeiten und die Arbeit an den Komponenten abzuschließen, die im Förderzeitraum zeitlich nicht mehr umgesetzt werden konnten. Das beinhaltet zunächst, die Verwaltung von Zugriffsrechten, Einladung von Teammitgliedern und Ende-zu-Ende-Verschlüsselung auf der Grundlage der Signature-Chain abzuschließen, sowie das Zusammenwirken der Komponenten und ihrer übertragbaren Daten und die Konflikt-Erkennung umzusetzen. Im Anschluss ist eine geschlossenen Beta-Testphase geplant, in der Fehlerbehebungen, Korrekturen und Anpassungen auf der Basis des User-Feedbacks stattfinden, bevor RoaringRoster offiziell über die App Stores und die Projektwebseite veröffentlicht wird. Für die Datensynchronisierung können die Teams entscheiden, ob sie den voreingestellten und von mir gehosteten Websocket-Server zur Datensynchronisierung nutzen wollen, oder diesen selbst hosten möchten.

Auch nach der Veröffentlichung beabsichtige ich die App um weitere Funktionen und Datenübertragungstechniken zu erweitern. Zunächst möchte ich die Arbeitszeiterfassung ergänzen unter Berücksichtigung des Arbeitszeitgesetzes. Es folgen Funktionen zur Reservierung von Ressourcen und Räumen, zur Benachrichtigungen via Push-Notification über Ereignisse, die die Nutzer:in betreffen, sowie Team-Kalender und gemeinsame, kollaborative Aufgabenlisten. Ich plane dezentrale Techniken zur Peer-to-Peer-Datenübertragung via Bluetooth oder WebRTC als Network-Adapter zu ergänzen, sowie Unterstützung für WebDAV als zusätzliche serverbasierte Alternative.

Um die Weiterentwicklung der App und einen zuverlässigen, wirtschaftlichen Betrieb einschließlich Support dauerhaft über Förderungen hinaus finanzieren zu können, beabsichtige ich für die App-Nutzung ein Software-Abomodell einzuführen, wobei die Preisgestaltung noch unter Einbeziehung von Nutzer:innen-Feedback zu klären ist. Für den strukturellen und rechtlichen Rahmen der Software-Entwicklung und des Vertriebs möchte ich einen Kollektivbetrieb gründen, voraussichtlich in der Rechtsform einer Genossenschaft, um selbst im Team zu arbeiten und das Erfahrungswissen über Methoden zur Selbstorganisation zu vertiefen, sowie RoaringRoster für den eigenen Bedarf zu nutzen, wovon die Weiterentwicklung profitieren soll.

Durch die Projektarbeit konnte ich mein fachliches Wissen und Verständnis wesentlich vertiefen sowohl über die Anforderungen an kollaborative Dienst- und Abwesenheitsplanung mit den benötigten Workflows und Benutzeroberflächen einerseits, als auch über CRDT-Datenstrukturen und über Architekturen und technische Lösungen für dezentrale Authentifizierung und Datenübertragung anderseits. Doch die wichtigste Erkenntnis für mich ist, dass die Local-First-Technologien im Umfeld der Automerge-Community mittlerweile ein Entwicklungsstadium erreicht haben, in dem sie produktiv als technische Basis von Apps einsetzbar sind.

Kurze Darstellung der Arbeiten, die zu keiner Lösung geführt haben

Es gab keine Arbeiten, die zu keiner Lösung führten oder aus fachlichen Gründen nicht weiter verfolgt wurden. Wie in den Abschnitten „Ausführliche Darstellung der Ergebnisse” und „Kurze Erläuterung zur Einhaltung der Arbeits- und Kostenplanung” dargestellt, gab es zwar Arbeiten, die ich aus Zeitgründen nicht fertigstellen konnte, diese werde ich aber im Anschluss an den Förderzeitraum weiter verfolgen und umsetzen.

Kurze Angabe von Präsentationsmöglichkeiten für mögliche Nutzer

Projekt-Webseite: https://www.roaringroster.app
Code: https://github.com/roaringroster/webapp

Kurze Erläuterung zur Einhaltung der Arbeits- und Kostenplanung

Mehrere Meilensteine und Teilaufgaben beinhalteten Mehraufwand, der eine Anpassung der Planung erforderlich machte. Besonders aufwändig war die Konzeption und Umsetzung der Dienstplanung und der Abwesenheitsplanung: angefangen mit der Ermittlung der Anforderungen, der Begrenzung des Funktionsumfangs auf das, was im Projektzeitraum realistisch umsetzbar ist, dem Entwurf der Benutzeroberfläche unter Abwägung von Verständlichkeit, Effizienz und Einfachheit verschiedener Lösungsansätze für die Anwender:in, über die Umsetzung, dem Ausprobieren, Revidieren und Verfeinern der vielversprechendsten Ansätze bis hin zur Fehlerbehebung und dem integrierten Zusammenwirken der einzelnen Komponenten und ihrer Daten. Infolgedessen konnte die Bearbeitung der verfügbaren Arbeitszeiten der Teammitglieder und das Zusammenwirken der Komponenten und ihrer Daten zeitlich nicht mehr umgesetzt werden. Die in den Mehraufwand der Dienst- und Abwesenheitsplanung investierte Zeit fehlte auch für die Funktionen zur Verwaltung von Zugriffsrechten, zur Einladung neuer Teammitglieder, der Ende-zu-Ende-Verschlüsselung, der Konflikt-Erkennung, sowie für ein robustes User-Testing und die Vorbereitung einer Veröffentlichung in den App Stores.

Eine weitere Anpassung der Planung habe ich vorgenommen, als ich erkannte, dass ein personalisiertes Übersichts-Dashboard aus Nutzer:innen-Sicht sinnvoll ist, um dringenden Handlungsbedarf und relevante Informationen auf einen Blick zu erkennen, wie beispielsweise kurzfristige Krankmeldungen, Anfragen zum Schicht-Tausch oder die eigenen Arbeitszeiten der kommenden Woche. Ich entschied mich, die Umsetzung der ursprünglich geplanten Arbeitszeitkonto-Funktion zugunsten des Dashboards zurückzustellen und diese zu einem späteren Zeitpunkt nach der Förderung umzusetzen. Ausschlaggebend für diese Entscheidung war für mich, dass das Dashboard in der Integration mit den Komponenten der Dienst- und Abwesenheitsplanung einen Mehrwert bietet, der den der eigenständig nutzbaren Funktion zur Verwaltung von Arbeitszeitkonten übersteigt.

Kurze Darstellung von etwaigen Ergebnissen bei anderen Stellen

Einfluss auf meine Arbeit haben die Entwickler:innen der Open-Source-Community von Automerge und der Auth-Bibliothek durch die Weiterentwicklung dieser Komponenten. Ihre Entwicklung der Automerge Storage- und Network-Adapter führte dazu, meine Architektur für Datenspeicherung und -übertragung zu überdenken und ihre flexiblere Architektur mit einheitlichen Schnittstellen für die leichter wiederverwendbaren und aus der Community beigetragenen Storage- und Network-Adapter zu übernehmen. Dies ermöglicht mittelfristig eine effizientere Entwicklung von RoaringRoster, da sich Synergien mit der Community ergeben und ich weniger App-spezifischen, eingeschränkt wiederverwendbaren Programmcode schreiben muss.