Von der klickbaren PDF zur App

Wie ich ohne große Programmierkenntnisse 1Rettungsmittel gebaut habe

MARC-ANDRÉ HOFFMANN

Eine ehrliche Entwicklungsgeschichte aus Ausbildung, Rettungsdienst, Fehlermeldungen und sehr viel Ausprobieren.

1Rettungsmittel

Wie aus einer Idee aus der Ausbildung eine echte App wurde

Marc-André Hoffmann

Stand September 2026

Über den Autor und das Projekt

Marc-André Hoffmann ist Notfallsanitäter und Berufsfeuerwehrmann. Aus der praktischen Ausbildung heraus entstand die Idee zu 1Rettungsmittel: Übungen schneller vorbereiten, digitale Patienten nutzen und Abläufe so trainierbar machen, dass sie in der Praxis wirklich helfen.

Dieses E-Book erzählt, wie aus einer klickbaren PDF eine App wurde: mit Umwegen, Fehlermeldungen, Rückmeldungen aus der Ausbildung und Menschen, die aus einer Idee mehr gemacht haben als nur Code.

Inhalt

Kapitel 1. Am Anfang war es nur eine PDF

Die Idee für 1Rettungsmittel entstand nicht, weil ich unbedingt eine App entwickeln wollte. Ich hatte auch nie den großen Traum, Softwareentwickler zu werden. Eigentlich wollte ich nur ein Problem lösen, das mich in der Ausbildung immer wieder beschäftigt hatte.

Ich arbeite im Rettungsdienst und bin Notfallsanitäter. Gerade bei Übungen zu einem MANV, einem Massenanfall von Verletzten, gibt es viele gute analoge Hilfsmittel. Patientenkarten, Übungsmaterial, Verletzungsmuster und vorbereitete Szenarien funktionieren fachlich sehr gut.

Mein Problem war also nicht die Qualität. Mein Problem war der Aufwand. Wenn eine große Übung geplant wird, ist Vorbereitung völlig normal. Dann gibt es einen Termin, mehrere Ausbilder, Material und entsprechend Zeit. Aber Ausbildung findet nicht immer so statt.

Manchmal hat man noch eine Stunde. Manchmal möchte man mit einer kleinen Gruppe spontan etwas trainieren. Manchmal denkt man einfach: Eigentlich könnten wir jetzt noch eine kleine MANV-Lage machen, und dann beginnt die Vorbereitung. Material holen, Patientenkarten zusammenstellen, die Lage vorbereiten, Verletzungsmuster auswählen und überlegen, wie man später auswertet.

Aus einer spontanen Idee wird ziemlich schnell etwas für nächste Woche. Genau das störte mich. Ich wollte ein Werkzeug, das man

öffnen und praktisch sofort einsetzen kann. Lage auswählen, Patienten bereitstellen, loslegen: wenig Vorbereitung, wenig Material und trotzdem sinnvoll trainieren.

Ich suchte nach einer Lösung, die genau das konnte. Ich fand gute Systeme. Aber nichts, das genau meinem Gedanken entsprach. Also kam irgendwann dieser gefährliche Satz: Dann müsste man so etwas eigentlich selbst bauen. Das Problem daran war nur: Ich konnte keine Apps programmieren.

HTML, CSS und JavaScript spielten in meinem Leben nur eine kleine Rolle. Lange ist es her, dass ich in der Schule mal in einer AG war. Von Servern, Backends oder App Stores hatte ich ebenfalls keine Ahnung. Ich hatte eigentlich nur die Idee. Also fing ich mit dem an, was ich hinbekam. Mit einer PDF.

Nicht mit einer App, nicht mit einer Plattform, sondern mit einer klickbaren PDF. Heute würde man wahrscheinlich sagen, es war ein Prototyp. Damals war es für mich einfach eine PDF, auf der man Dinge anklicken konnte. Aber plötzlich existierte meine Idee nicht mehr nur in meinem Kopf. Ich konnte etwas zeigen. Ich konnte ausprobieren, ob der Ablauf überhaupt Sinn ergibt, und genau das war wahrscheinlich der wichtigste Schritt. Ich habe nicht gewartet, bis ich wusste, wie man eine professionelle App entwickelt. Ich habe einfach mit dem angefangen, was ich konnte.

Im September 2024 waren meine Frau Carina und ich in Österreich im Urlaub. Eigentlich sollte man im Urlaub abschalten. Ich tat ungefähr das Gegenteil. Beim Essen fing ich natürlich wieder mit der Idee an. Ich hatte mein Bier, Carina ihren Cocktail, und je länger der Abend wurde, desto besser fand ich meinen Plan. Zumindest in meinem Kopf.

Carina saß mir gegenüber und musste sich anhören, warum diese digitale Lösung für die Ausbildung aus meiner Sicht gerade eine

ziemlich gute Idee war, und natürlich blieb es nicht bei einem kurzen Satz. Ich erklärte, und spätestens beim nächsten Bier war ich überzeugt: Das wird richtig gut. Meine Frau sagte nicht: Lass den Quatsch.

Sie stellte Fragen. Was bringt das konkret? Wie soll das funktionieren? Was wäre der Vorteil? Und irgendwann sinngemäß: Wenn du glaubst, dass das für die Ausbildung etwas bringt, dann mach es. Das war damals wichtiger, als es vielleicht klingt. Denn es gab nichts, was bewiesen hätte, dass daraus jemals etwas wird. Keine Kunden. Keine App. Keine Downloads. Keine Referenzen. Nur eine klickbare PDF und jemanden, der ziemlich fest daran glaubte, dass daraus mehr werden könnte, und genau das passierte.

Sobald die PDF funktionierte, kam die nächste Frage. Kann man das nicht besser auf dem Smartphone oder Tablet darstellen? Kann man Patienten digital aufrufen? Kann man QR-Codes nutzen? Kann das Ganze auch offline funktionieren? Und irgendwann: Kann man daraus nicht irgendwie eine richtige App machen? Damit begann der Teil der Geschichte, bei dem ich wahrscheinlich vernünftigerweise hätte sagen können: Keine Ahnung davon. Lass lieber jemanden ran, der das gelernt hat. Stattdessen machte ich weiter. Rückblickend war das entweder mutig oder naiv. Wahrscheinlich beides.

Aber eines war mir ziemlich früh klar: Das Problem aus der Ausbildung war real, und solange ich glaubte, dass meine Idee dieses Problem lösen konnte, wollte ich herausfinden, wie weit ich damit komme.

Kapitel 2. Kann man daraus nicht irgendwie eine App machen?

Nachdem die klickbare PDF existierte, war relativ schnell klar, dass sie nicht das Ende sein würde.

Ich wollte mehr. Die Anwendung sollte reagieren. Patienten sollten unterschiedliche Informationen haben. Buttons sollten tatsächlich etwas auslösen, und irgendwann sollte sich das Ganze wirklich wie eine Anwendung anfühlen. Damit kam ich zu einem Bereich, von dem ich bis dahin wenig Ahnung hatte. Programmierung.

Ich hatte keinen klassischen Einstieg. Kein Lehrbuch. Keinen Kurs. Keinen Dozenten. Ja, ich war mal in einer AG, aber das ist über 20 Jahre her! Dafür hatte ich ziemlich viele Fragen und KI. Also begann ich genau dort. Wie baut man eine Webseite, die sich wie eine App anfühlt? Wie funktioniert ein Button? Wie wechselt man Inhalte? Wie speichert man Daten? Und vor allem: Warum geht das nicht? Dieser Satz sollte mich noch lange begleiten.

Die ersten Begriffe waren HTML, CSS und JavaScript. HTML bildet vereinfacht gesagt die Struktur einer Webseite. CSS bestimmt, wie sie aussieht. JavaScript sorgt dafür, dass etwas passiert. So würde ich das heute erklären. HTML war relativ dankbar. Man schrieb etwas und konnte danach etwas sehen. CSS machte ebenfalls Spaß. Farben ändern. Abstände. Schriften. Buttons. Man sah sofort ein Ergebnis.

JavaScript war der Punkt, an dem es wirklich interessant wurde. Plötzlich konnte ein Button etwas auslösen. Ein Patient konnte geöffnet werden. Inhalte konnten sich verändern, und zum ersten

Mal fühlte sich das Ganze nicht mehr wie eine klickbare Datei an. Es fühlte sich nach Software an.

Das Problem war nur: Ich verstand längst nicht immer, warum etwas funktionierte. Wenn KI mir 30 Zeilen Code gab und danach genau das passierte, was ich wollte, war mein erster Gedanke selten: Jetzt analysiere ich erst mal ausführlich die Logik. Mein Gedanke war eher: Geil. Nächste Funktion. Das rächte sich natürlich.

Aus kleinen Dateien wurden größere. Aus 100 Zeilen wurden irgendwann mehrere hundert, und plötzlich sucht man in einem Code mit 900 Zeilen nach einem Fehler. Man ändert etwas. Nichts geht. Man ändert es zurück. Jetzt geht noch weniger. Dann stellt man irgendwann fest: Es fehlte ein Komma. Ein Komma!

Andere Menschen sehen ein Satzzeichen. Ich sehe darin inzwischen potenziell mehrere Stunden verlorene Lebenszeit. Auch kleine optische Änderungen konnten interessant werden. Ich wollte zum Beispiel nur eine Farbe ändern. Eigentlich kein Drama. Visual Studio Code öffnen. Farbwert ändern. Speichern. Neu laden, und plötzlich funktioniert etwas anderes nicht mehr.

Das sind die Momente, in denen man ernsthaft darüber nachdenkt, ob Computer manchmal einfach persönlich beleidigt sind. Natürlich gab es fast immer eine technische Erklärung. Eine Klammer. Ein Zeichen. Etwas an der falschen Stelle. Aber der Gedanke blieb: Ich wollte doch nur die Farbe ändern. In dieser Phase tauchte auch der Begriff PWA auf.

PWA steht für Progressive Web App. Vereinfacht gesagt ist das eine Webanwendung, die sich in vielen Punkten wie eine klassische App verhalten kann. Sie kann zum Beispiel auf einem Tablet oder Smartphone genutzt, auf dem Startbildschirm abgelegt und teilweise auch offline betrieben werden.

Offline war für mich sofort interessant. Eine Trainingsanwendung für Rettungsdienst und Feuerwehr sollte schließlich nicht davon abhängig sein, dass in jedem Gebäude perfektes WLAN vorhanden ist. Also dachte ich: Dann machen wir eben eine PWA. Das klingt im Nachhinein deutlich souveräner, als es damals war.

Ich wusste nicht, wie man eine PWA baut. Ich wusste nur, dass sie offenbar das konnte, was ich brauchte. Also wieder KI fragen. Welche Dateien brauche ich? Was ist ein Manifest? Was ist ein Service-Worker? Wie funktioniert Offline-Speicherung?

Ein Service-Worker ist vereinfacht gesagt ein Teil der Anwendung, der im Hintergrund arbeiten und unter anderem Dateien zwischenspeichern kann. Das klingt harmlos. Bis man merkt, dass der Zwischenspeicher auch dafür sorgen kann, dass man nach einer Änderung immer noch die alte Version sieht.

Ich änderte etwas. Lud neu. Alte Version. Noch mal. Immer noch die alte Version. Irgendwann fing ich ernsthaft an zu glauben, dass ich vielleicht die falsche Datei bearbeite. Also öffnete ich wieder den Code, änderte erneut, speicherte und lud neu. Nichts, und irgendwann war ich so weit, dass ich einzelne Seiten einfach noch mal neu gebaut habe, weil ich dachte: Dann muss es ja jetzt funktionieren.

Tat es nicht.

Irgendwann kam dann die Erkenntnis, die mir wahrscheinlich einige Stunden meines Lebens hätte sparen können: Der Service-Worker musste aktualisiert werden. Natürlich. Woher sollte ich das bitte wissen? Für jemanden, der sich damit auskennt, ist das vermutlich völlig selbstverständlich. Für mich war es das nicht. Ich hatte bis kurz davor nicht einmal gewusst, was ein Service-Worker überhaupt ist.

Also wieder etwas gelernt. Ziemlich teuer bezahlt, zumindest in Lebenszeit. Aber genau so lief es ständig. Ein Problem, Frust, noch ein Versuch, dann irgendwann die Lösung, und wieder hatte ich ein kleines Stück mehr verstanden. Der Glaube daran, dass aus der Sache etwas werden kann, war trotzdem weiter da, und mein Ehrgeiz auch. Mit der Zeit passierte aber etwas Wichtiges.

Ich kopierte nicht mehr nur Code und hoffte, dass er irgendwie funktioniert. Ich fing an, ihn wirklich zu lesen. Ich erkannte Funktionen wieder, verstand langsam, wo Daten gespeichert wurden, und konnte immer besser einschätzen, was eine Änderung eigentlich bewirken sollte. Genau an dieser Stelle lernte ich allerdings auch, dass das Wort „müsste“ in der Softwareentwicklung ziemlich gefährlich ist.

Eigentlich müsste das funktionieren. Eigentlich habe ich doch nur diese eine Sache geändert. Eigentlich kann das damit gar nichts zu tun haben. Wenn man einen dieser Sätze denkt, ist meistens der Moment gekommen, sich erst mal einen Kaffee zu holen.

Aber genau durch diese Probleme lernte ich. Nicht nach Lehrplan. Sondern immer dann, wenn ich etwas brauchte. Ich wollte nie programmieren lernen, um Programmierer zu sein. Ich wollte ein Werkzeug für die Ausbildung bauen, und um dieses Werkzeug zu bauen, musste ich eben lernen, wie man programmiert. Das machte einen riesigen Unterschied.

Ich musste nicht alles verstehen, bevor ich anfangen durfte. Ich konnte das nächste Problem lösen. Dann das nächste. Dann noch eins, und irgendwann war aus der PDF tatsächlich eine erste Anwendung geworden. Noch nicht schön. Noch nicht perfekt. Aber real genug, dass ich dachte: Das könnte wirklich funktionieren.

Kapitel 3. Plötzlich brauchte meine Idee eine Infrastruktur

Am Anfang war mein Projekt überschaubar. Ein paar Dateien: HTML, CSS, JavaScript, ein bisschen KI und viel Hoffnung. Solange alles nur auf meinem Rechner lag, war das irgendwie beherrschbar. Datei öffnen, ändern, speichern, testen. Problematisch wurde es erst, als ich irgendwann nicht mehr genau wusste, welche Datei eigentlich die richtige war.

Wer jemals ohne Versionsverwaltung an einem Projekt gearbeitet hat, kennt vermutlich Dateinamen wie final.html, final_neu.html, final_neu2.html, final_wirklich.html und irgendwann wahrscheinlich final_jetzt_wirklich.html. Ich hatte also durchaus ein System. Nur kein besonders gutes.

Je größer das Projekt wurde, desto gefährlicher wurde diese Arbeitsweise. Mehr Dateien, mehr Funktionen, mehr Patienten, mehr Logik, und irgendwann die berechtigte Angst: Wenn ich hier jetzt etwas kaputtmache, habe ich hoffentlich noch irgendwo eine funktionierende Version. Damit kam GitHub ins Spiel.

GitHub ist eine Plattform, auf der Code gespeichert und Änderungen nachvollziehbar verwaltet werden können. Ich kannte den Namen. Benutzt hatte ich es nicht. Plötzlich begegneten mir Begriffe wie Repository, Commit, Push, Pull und Merge. Ich fragte mich kurz, warum Entwickler nicht einfach normale Wörter benutzen. Aber gut. Ich lernte es.

Nicht weil Versionsverwaltung meine große Leidenschaft wurde. Sondern weil ich mein eigenes Chaos überleben wollte, und das funktionierte.

Langsam wurde mein Projekt strukturierter. Änderungen konnten nachvollzogen werden. Funktionierende Stände ließen sich besser sichern, und die Wahrscheinlichkeit sank, dass eine Datei namens final_wirklich2 mein wichtigstes Backup war. Dann kam Hosting.

Eine Anwendung, die nur auf meinem Rechner funktioniert, bringt in einer Ausbildung wenig. Sie musste erreichbar sein. Also kamen Plattformen wie Netlify dazu. Code hochladen, Anwendung veröffentlichen, und plötzlich konnte ich jemandem einen Link schicken. Auf einem anderen Gerät erschien das, was ich entwickelt hatte. Das war ein besonderer Moment.

Bis dahin war das Projekt in meiner kleinen technischen Welt geblieben. Jetzt konnte es jemand anderes öffnen. Natürlich bedeutete das auch: Jetzt können andere sehen, wenn etwas nicht funktioniert, und genau damit entstand eine neue Kategorie von Problemen. Lokal läuft es. Online nicht. Ein Klassiker.

Dann sucht man. Pfad falsch? Dateiname? Großschreibung oder Kleinschreibung? Deployment? Cache? Oder hatte ich einfach irgendwo Mist gebaut? Die letzte Möglichkeit sollte man nie komplett ausschließen. Dazu kamen Domain, HTTP, HTTPS und immer mehr Fragen zur technischen Struktur.

Ich musste erst mal verstehen, warum mir eine Seite über HTTP teilweise nicht richtig angezeigt wurde, über HTTPS aber schon. Dann bekam ich ständig Hinweise, dass HTTP unsicher sei und heute eigentlich HTTPS genutzt werden sollte. Meine erste Reaktion war ungefähr: Wenn HTTP so unsicher ist, warum schaltet man das dann nicht einfach komplett ab?

Für jemanden mit technischem Hintergrund ist das wahrscheinlich alles logisch. Für mich war es einfach das nächste Thema, in das ich mich einarbeiten musste, und natürlich liefen solche Erkenntnisse selten völlig entspannt ab.

Es gab Tage, da habe ich meinen Rechner ernsthaft beleidigt. Dabei konnte der nun wirklich nichts dafür. Im Grunde war er einfach nur mein stiller Mitarbeiter, der sich meinen Frust anhören musste, während ich versuchte zu verstehen, warum etwas schon wieder nicht so funktionierte, wie ich es mir vorgestellt hatte.

Eigentlich hätte ich diese Zeit einfach mitfilmen sollen. Nur ich, mein Rechner und mein langsam steigender Puls. Wahrscheinlich hätte ich damit im Internet zumindest ein paar Leute zum Lachen gebracht, und das kann ich heute übrigens auch selbst. Wenn ich daran zurückdenke, wie ernst ich manche Fehlermeldung genommen habe, muss ich inzwischen eher über mich lachen als über den Rechner. Aber es ging weiter, und irgendwann merkte ich: Ich baue längst nicht mehr nur eine Oberfläche. Ich baue Infrastruktur.

Wo liegt mein Code? Wie veröffentliche ich Änderungen? Wie sichere ich funktionierende Versionen? Wie spiele ich Updates ein? Was passiert, wenn etwas schiefgeht? Diese Dinge sieht später kein Nutzer. Kein Mensch öffnet eine App und sagt: Wow. Wirklich hervorragend gelöstes Versionsmanagement. Aber wehe, im Hintergrund ist es schlecht gelöst. Dann merken es plötzlich alle.

Mit dem wachsenden Projekt änderte sich auch mein Denken. Ich fragte nicht mehr nur: Wie bekomme ich diese Funktion hin? Sondern zunehmend: Wie baue ich das so, dass ich in drei

Monaten noch verstehe, was ich da gemacht habe? Das gelang nicht immer.

Es gab durchaus Situationen, in denen ich später meinen eigenen Code ansah und dachte: Welcher Idiot hat das so gebaut? Dann fiel mir ein: Ach ja. Ich. Aber genau diese Phase war wichtig. Ich lernte, dass Software nicht nur aus dem besteht, was man sieht, und ich lernte, dass Sicherung, Struktur und Wiederherstellung keine Themen sind, um die man sich irgendwann später kümmern sollte.

Spätestens wenn Wochen und Monate Arbeit in einem Projekt stecken, bekommt ein funktionierendes Backup plötzlich einen ganz anderen Charme. Gleichzeitig tauchte langsam die nächste technische Welt auf. Backend. Das Frontend ist vereinfacht gesagt der Teil, den ein Nutzer sieht und bedient. Das Backend arbeitet im Hintergrund. Dort können zum Beispiel Daten gespeichert, Lizenzen geprüft oder Anfragen verarbeitet werden.

Am Anfang brauchte ich das kaum. Später sollte genau dieser Bereich immer wichtiger werden. Aber zunächst ging es um etwas anderes. Ich wollte meine digitalen Patienten nicht mehr nur über Menüs öffnen. Ich brauchte eine schnelle Verbindung zwischen realer Übung und digitaler Anwendung, und dafür bot sich etwas an, das später noch ziemlich häufig in meinem Projekt auftauchen sollte. QR-Codes.

Kapitel 4. Ein Patient bekommt einen QR-Code

Die technische Grundlage war irgendwann weit genug. Die Anwendung lief. Sie war erreichbar. Sie konnte auf verschiedenen Geräten geöffnet werden. Jetzt kam eine sehr praktische Frage: Wie kommt ein Teilnehmer schnell zum richtigen Patienten?

Ich wollte nicht, dass jemand während einer Übung lange in Menüs sucht. Vor ihm liegt ein Patient. Er soll direkt mit diesem Patienten arbeiten können. Die Lösung lag eigentlich nahe. QR-Codes. Ein Patient, ein Code, scannen, Patient öffnen, fertig. Zumindest in meiner Vorstellung.

Ein QR-Code ist ein zweidimensionaler Code, der mit einer Kamera gelesen werden kann und zum Beispiel Informationen oder einen Link enthält. Für 1Rettungsmittel passte das perfekt. Ein Ausdruck oder eine Kennzeichnung beim Patienten konnte direkt auf dessen digitale Darstellung führen. Damit verband sich die reale Übung mit der Anwendung. Also begann ich, einen Scanner einzubauen, und natürlich entstanden sofort neue Probleme. Kamerazugriff. Berechtigungen. Unterschiedliche Geräte. Scanner-Bibliotheken.

Manchmal erkannte die Kamera den Code sofort. Manchmal nicht. Manchmal wurde derselbe Code mehrfach erfasst. Manchmal passierte gar nichts. Wieder die üblichen Fragen. Code? Kamera? Browser? Gerät? Oder ich? Aber der grundsätzliche Ablauf funktionierte, und das war ein wichtiger Moment. Zum ersten Mal fühlte sich das Ganze wirklich nach

einem Trainingssystem an. Dann kam die nächste Frage: Was zeigt der digitale Patient eigentlich?

Nicht nur Patient 1. Sondern Verletzung. Zustand. Vitalparameter. Informationen. Sichtung, und möglichst so, dass daraus ein realistisches Training entsteht. Damit wurde aus Programmierung plötzlich auch Didaktik.

Was darf der Teilnehmer sofort wissen? Welche Informationen soll er erst bei einer Untersuchung bekommen? Wie verhindere ich, dass die richtige Sichtungskategorie schon beim ersten Blick offensichtlich ist? Wie komplex darf ein Patient werden, bevor die Bedienung darunter leidet?

Das war genau die Stelle, an der mein beruflicher Hintergrund wichtig wurde.

Technisch musste ich alles neu lernen. Aber das Ausbildungsziel kannte ich. Ich wusste, welche Entscheidungen die Teilnehmer treffen sollten. Ich wusste, welche Informationen relevant waren, und ich wusste, dass eine technisch beeindruckende Funktion vollkommen nutzlos sein kann, wenn sie in der Ausbildung keinen Mehrwert bringt. Dann kam das Thema Bilder.

Reiner Text funktionierte. Aber Bilder machten die Patienten greifbarer, und damit begann die Karriere meiner Familie als Patientendarsteller. Professionelle Models hatte ich nicht. Dafür hatte ich Familie, Freunde und eine Garage. Also wurden Verletzungen dargestellt. Menschen geschminkt. Positionen ausprobiert. Fotos gemacht.

Carina gehörte ziemlich früh zu denjenigen, die sich für das Projekt hinlegen und zum Opfer machen lassen mussten. Ob das Bestandteil unserer Eheplanung gewesen war, weiß ich nicht mehr. Wahrscheinlich nicht. Später kamen weitere

Familienmitglieder und Freunde dazu. Nicole, Kollegin und gute Freundin, half ebenfalls. Wenn ich selbst auf einem Bild das Opfer war, musste schließlich jemand die Kamera bedienen.

Andere Menschen machen Familienfotos. Wir produzierten Verletzungsmuster. Normal. Mit den Bildern und den QR-Codes konnte ich immer mehr Patienten aufbauen, und je mehr Patienten entstanden, desto stärker wurde die Anwendung. Gleichzeitig wuchs aber auch die Versuchung, ständig neue Dinge einzubauen.

Wenn Patienten digital sind, könnten sie sich doch auch verändern. Wenn jemand zu spät handelt, könnte sich der Zustand verschlechtern. Wenn ein Patient gesichtet wurde, könnte man das speichern. Später vielleicht auswerten. Vielleicht könnte man Patienten übergeben. Vielleicht Patientenablagen trainieren.

Es entstand langsam eine Liste von Ideen, die deutlich schneller wuchs als meine verfügbare Zeit, und da begann eine Angewohnheit, die mich lange begleiten sollte: Ich teste nur noch kurz etwas. Kurz ist in der Softwareentwicklung ein interessantes Wort. Manchmal bedeutet es fünf Minuten. Manchmal schaut man danach auf die Uhr und es ist Mitternacht. Aber die Richtung stimmte.

Aus der klickbaren PDF war eine Anwendung geworden, die reale Patienten über QR-Codes mit digitalen Informationen verband. Das war ziemlich nah an dem, was ich am Anfang gesucht hatte. Jetzt fehlte nur noch etwas Entscheidendes. Andere Menschen mussten damit arbeiten. Nicht ich. Nicht jemand, dem ich direkt erklären konnte, was er anklicken soll. Sondern echte Nutzer.

Kapitel 5. Zuhause lief alles. Natürlich.

Irgendwann muss man bei einem Projekt aufhören, nur selbst darauf herumzuklicken.

Man braucht echte Nutzer. Menschen, die nicht wissen, was man sich bei einem Button gedacht hat. Menschen, die Dinge in einer Reihenfolge machen, die man niemals vorgesehen hat, und vor allem Menschen, die einem ohne Rücksicht auf die vielen Arbeitsstunden sagen: Das funktioniert nicht. Genau diesen Punkt erreichte ich mit 1Rettungsmittel.

Zuhause hatte ich getestet. Sehr viel. Patienten geöffnet. QR-Codes gescannt. Szenarien gestartet. Funktionen wiederholt. Wenn etwas nicht funktionierte, suchte ich den Fehler. Wenn es funktionierte, testete ich es noch mal. Irgendwann hatte ich das Gefühl: Das läuft, und dann kam der reale Unterricht.

Ich hatte das Glück, 1Rettungsmittel bei meiner Arbeit in der Berufsfeuerwehr ausprobieren zu dürfen. Das war enorm wichtig. Denn zuhause kann man seine eigene Software erstaunlich erfolgreich testen. Man weiß schließlich, wie sie gedacht ist. Man klickt automatisch an den richtigen Stellen. Man kennt kleine Eigenheiten. Man weiß, was man besser nicht macht. Andere Menschen haben diese Vorteile nicht. Die nehmen ein Tablet und benutzen es. Unverschämt eigentlich.

Plötzlich wird zweimal geklickt. Zurückgegangen. Etwas in einer anderen Reihenfolge geöffnet. Ein QR-Code unter anderen Bedingungen gescannt, und natürlich passiert dann etwas, das zuhause niemals passiert ist. Fehler. Ich hasste diese Momente.

Nicht, weil jemand einen Fehler gefunden hatte. Sondern weil ich ihn vorher nicht gefunden hatte.

Man steht im Unterricht daneben und denkt: Warum jetzt? Gestern hat das zehnmal funktioniert. Nach außen versucht man möglichst ruhig zu wirken. Innerlich läuft eine etwas andere Unterhaltung. Liegt es am Tablet? An der Kamera? Am Cache? Am Code? An der Verbindung? Warum genau jetzt? Und natürlich der Klassiker: Bei mir ging es eben noch.

Noch intensiver wurde es, als andere das System benutzten, ohne dass ich dabei war. Das war noch mal eine ganz andere Nummer. Solange ich danebenstand, konnte ich eingreifen. Da musst du einmal zurück. Lade kurz neu. Nimm diesen Button. Wenn ich nicht dabei war, gab es diese Rettungsleine nicht. Dann musste die Anwendung für sich selbst funktionieren. Ohne mich. Ohne Erklärung. Ohne improvisierten Support neben dem Teilnehmer. Genau das war heftig. Denn damit stellte sich erstmals wirklich die Frage: Ist die App intuitiv? Oder kann nur ich sie bedienen, weil ich sie selbst gebaut habe?

Gerade bei den ersten externen Testungen, zum Beispiel in Bayern oder Dresden, merkte ich, wie nervös mich das machte. Meistens hatte man schon vorher per WhatsApp Kontakt. Dann wusste ich: Heute wird getestet. Natürlich hielt ich es nicht lange aus und schrieb irgendwann: „Und, wie lief es?“

Dann kamen diese kleinen Haken im Chat. Erst einer. Dann zwei. Dann wurden sie blau. Gelesen, und dann kam erst mal nichts. Genau in diesem Moment begann mein Kopf zu arbeiten. Warum schreibt er oder sie nicht zurück? War es scheiße? Ist etwas abgestürzt? Hat gar nichts funktioniert? Haben die gerade beschlossen, dass sie die App nie wieder sehen wollen?

Wahrscheinlich saß auf der anderen Seite einfach jemand im Unterricht oder hatte gerade keine Zeit. Für mich fühlten sich diese paar Minuten trotzdem teilweise an wie eine komplette Fehleranalyse ohne irgendeine Fehlermeldung. Erst wenn dann irgendwann die Antwort kam, war ich wieder etwas entspannter.

Diese Situationen haben mir ziemlich deutlich gezeigt, wie sehr ich inzwischen an dem Projekt hing. Es war längst nicht mehr nur Code auf meinem Rechner. Andere Menschen benutzten es, und plötzlich war mir sehr wichtig, was bei ihnen passierte. Natürlich kamen Fehler zurück. Aber es kamen auch Verbesserungsvorschläge.

Kolleginnen und Kollegen sagten: Warum machst du das nicht so? Könnte der Ablauf hier nicht einfacher sein? Müsste diese Information nicht früher kommen? Teilweise hatte ich Stunden in eine Funktion gesteckt. Dann schaut jemand fünf Sekunden darauf und sagt: „Das würde ich anders machen."

Der erste innere Gedanke ist ungefähr: Du weißt gar nicht, wie lange ich daran gesessen habe. Der zweite Gedanke ist deutlich hilfreicher: Mist. Er hat recht. Genau dadurch wurde das System besser. In solchen Phasen war ich auch ziemlich dankbar dafür, Praxisanleiter zu sein und über die Jahre viele Lehrgänge rund um Kommunikation besucht zu haben. Das hilft nämlich, Rückmeldungen einzuordnen.

Es gibt konstruktive Kritik, mit der man wirklich etwas anfangen kann, und es gibt Sätze wie: „Ich habe zwar keine Ahnung, aber das müsste anders funktionieren." Solche Aussagen konnte ich irgendwann ziemlich gut einordnen und eher belächeln, statt mich darüber aufzuregen. Denn nicht jede Rückmeldung ist automatisch richtig, nur weil sie ausgesprochen wird. Entscheidend ist, ob dahinter ein echtes Problem steckt, und

mindestens genauso wichtig wie die Kritik war etwas anderes. Die Rückendeckung.

Meine Kolleginnen und Kollegen fanden Fehler. Aber sie sahen auch den Nutzen. Sie verstanden, welches Problem das System lösen sollte. Weniger Vorbereitung. Schneller anfangen. Mehr Möglichkeiten in der Ausbildung. Das war für mich enorm wichtig. Denn es gab natürlich auch Momente, in denen ich mich fragte, warum ich mir das eigentlich antue. Wenn man nachts zum dritten Mal dieselbe Funktion umbaut, bekommt eine klickbare PDF plötzlich wieder einen gewissen Charme.

Dann kommt aber jemand und sagt: Wenn das stabil läuft, ist das richtig gut. Oder: Das könnte uns in der Ausbildung wirklich helfen, und schon sitzt man am nächsten Abend wieder am Rechner. Nicht weil das System perfekt ist. Sondern weil man merkt: Es lohnt sich, es besser zu machen.

Diese ersten Tests waren deshalb viel mehr als technische Prüfungen. Sie waren eine Realitätskontrolle. Funktioniert meine Idee nur in meinem Kopf? Oder auch dann, wenn andere damit arbeiten? Immer öfter lautete die Antwort: Ja. Noch nicht perfekt. Aber ja. Damit war der nächste Schritt eigentlich unausweichlich. 1Rettungsmittel musste raus aus meinem direkten Arbeitsumfeld.

Ich brauchte Menschen aus dem Bereich, die das System fachlich beurteilen konnten und mir trotzdem ehrlich sagen würden, wenn etwas nicht funktioniert oder keinen Sinn ergibt. Viele von ihnen kannten mich zwar bereits aus der Arbeit, aber genau das war für mich eher ein Vorteil. Sie wussten, woher die Idee kam, kannten die Praxis und hatten keinen Grund, mir irgendetwas schönzureden, und genau dort sollte das NAZ Goslar eine entscheidende Rolle spielen.

Kapitel 6. Als plötzlich andere an die Idee glaubten

Bis dahin hatte ich 1Rettungsmittel hauptsächlich in meinem direkten Umfeld getestet. Bei der Berufsfeuerwehr, mit Kolleginnen und Kollegen, mit Menschen, die mich kannten. Diese Rückmeldungen waren enorm wichtig. Aber irgendwann wollte ich wissen: Funktioniert die Idee auch außerhalb meiner eigenen Welt? Denn bei Kollegen bleibt immer ein kleiner Zweifel. Finden die das wirklich gut? Oder denken sie einfach: Lass ihn mal machen. Der beruhigt sich irgendwann wieder.

Ich brauchte Menschen aus der Ausbildung, die mir nichts schuldeten. Menschen, die keinen Grund hatten, meine Idee aus Höflichkeit gut zu finden, und genau an dieser Stelle wurde das Notfallmedizinische Ausbildungszentrum in Goslar wichtig.

Das NAZ Goslar war von Anfang an bereit, sich mit dem System zu beschäftigen. Nicht erst dann, als alles fertig und schön verpackt war. Sondern während der Entwicklung. Damit passierte etwas Neues. 1Rettungsmittel wurde nicht mehr nur von Menschen ausprobiert, bei denen ich im Zweifel danebenstehen konnte. Die Anwendung ging raus. Andere nutzten sie. Andere bewerteten sie, und irgendwann nutzten sie sie sogar, während ich überhaupt nicht dabei war.

Das war für mich noch mal eine andere Nummer. Denn solange ich danebenstehe, kann ich im Zweifel eingreifen. Wenn man nicht dabei ist, fällt das alles weg. Dann muss das Produkt alleine klarkommen. Genau das machte diese Tests so wertvoll, und ehrlich gesagt auch ziemlich nervenaufreibend.

Wenn ich wusste, dass irgendwo mit 1Rettungsmittel gearbeitet wurde, war ich natürlich neugierig. Hat alles funktioniert? Gab es Fehler? War etwas unverständlich? Ist jemand irgendwo hängen geblieben? Und natürlich kamen Rückmeldungen. Manche waren positiv. Andere bedeuteten Arbeit. Beides war wichtig. Denn ein externer Nutzer sieht Dinge, die man selbst irgendwann nicht mehr sieht. Ich war inzwischen tief im System. Für mich war logisch, wo welcher Button sitzt. Ein neuer Nutzer weiß davon nichts, und genau das ist gut.

Eine Anwendung ist nicht intuitiv, weil der Entwickler sie versteht. Sie ist intuitiv, wenn jemand sie versteht, der sie nicht gebaut hat. Das NAZ Goslar half mir dabei enorm. Es wurde getestet. Es kamen Fragen, Hinweise und Verbesserungsvorschläge. In dieser Zeit war auch Sven von Atem-Life wichtig.

Mit ihm war ich von Anfang an immer wieder im Austausch. Am Anfang war er durchaus skeptisch. Nicht ablehnend, eher so: Zeig mal, ob das wirklich trägt. Genau solche Menschen braucht man. Keine Dauerbegeisterung, die alles abnickt, sondern jemanden, der fragt, prüft und ehrlich sagt, wenn etwas noch nicht rund ist.

Mit der Zeit merkte ich, dass aus dieser Skepsis echtes Interesse wurde, und irgendwann auch so etwas wie Anerkennung. Sven wurde für mich immer mehr zu einem guten Ratgeber, gerade wenn ich bei einer Frage festhing oder jemanden brauchte, der nicht nur meine Begeisterung hörte, sondern auch die praktische Seite sah, und irgendwann merkte ich: Die Idee funktioniert nicht nur in meinem Kopf. Andere sehen darin tatsächlich einen Nutzen. Später wurde das NAZ Goslar mein erster Kunde.

Natürlich ging es auch ums Geld. Zum ersten Mal hatte jemand für etwas bezahlt, das ich selbst entwickelt hatte. Aber viel

wichtiger war die Bedeutung dahinter. Bis dahin hatte ich vor allem gehört: Das könnte funktionieren. Mach weiter. Das ist interessant.

Jetzt sagte eine externe Einrichtung im Grunde: Das ist uns etwas wert. Lob ist unverbindlich. Ein Kauf ist eine Entscheidung. Damit wurde aus meiner Idee endgültig ein Produkt, und gleichzeitig entstand eine neue Frage. Eine ziemlich schwierige sogar. Wenn jemand dafür bezahlen möchte, was soll es eigentlich kosten?

Kapitel 7. Was ist meine eigene Software eigentlich wert?

Diese Frage fand ich teilweise schwieriger als das Programmieren. Was kostet meine eigene Software eigentlich? Und natürlich dachte ich wieder viel zu groß. Ich wollte direkt für jeden irgendeine passende Lösung haben. Drei Stunden Zugriff. Zwölf Stunden. Vierundzwanzig Stunden. Eine Woche. Ein Monat. Am besten alles, und natürlich sollte das nicht einfach nur irgendwie funktionieren, sondern möglichst komplett automatisch.

Jemand klickt auf der Homepage auf ein Angebot, bezahlt über PayPal, bekommt sofort seinen Zugang, die Laufzeit startet automatisch und am Ende wird wieder gesperrt. Genau so hätte ich mir das selbst irgendwo gewünscht. Also baute ich es. Zahlungsabläufe, Aktivierung, Laufzeiten, Prüfungen, Fehlerfälle. Alles musste fertig sein, bevor überhaupt klar war, ob das irgendjemand wirklich so braucht. Ich wollte Perfektion. Heute denke ich mir: Leute, ganz ehrlich, was war denn da los?

Wenn mir jetzt jemand sagt: „Ey, genau das war wichtig“, dann nehme ich alles zurück. Aber rückblickend war ich da einfach ein bisschen lost. Ich hatte technisch ein ziemlich gutes System gebaut, nur eben für ein Problem, das meine eigentliche Zielgruppe so gar nicht hatte.

Eine Rettungsdienstschule möchte vor einer Ausbildung nicht überlegen, ob sie heute drei Stunden, zwölf Stunden oder sieben Tage Zugriff braucht. Sie möchte im Grunde wissen: Wir haben das System, wir können es nutzen, fertig. Genau da habe ich verstanden, dass technisch gut gelöst noch lange nicht bedeutet, dass etwas als Produkt gut gelöst ist, und ja, das tat weh, weil ich genau wusste, wie viele Stunden da drinsteckten. Aber es war eine wichtige Lektion.

Heute würde ich viel früher fragen: Wie wollt ihr das eigentlich nutzen? Damals habe ich oft erst gebaut und danach gemerkt, was die Kunden wirklich brauchen. Mit der Zeit änderte sich dadurch auch mein Blick auf den Preis. Ich fragte nicht mehr: Wie viele Stunden Zugriff verkaufe ich? Sondern: Welchen wirtschaftlichen Nutzen hat die Software?

Eine klassische MANV-Ausbildung kann erheblichen Aufwand verursachen. Je nach Lage sitzt ein Ausbilder vielleicht zwei, drei oder auch fünf Stunden an Vorbereitung und Nachbereitung. Während einer größeren Durchführung sind möglicherweise zwei oder drei Ausbilder gebunden. Material muss vorbereitet werden. Hinterher muss ausgewertet werden.

All diese Zeit kostet Geld. Personal kostet Geld. Vorbereitung kostet Geld. Wenn 1Rettungsmittel dafür sorgt, dass eine Lage mit deutlich weniger Vorbereitung durchgeführt werden kann, entsteht daraus ein realer wirtschaftlicher Nutzen. Der Kunde bezahlt nicht dafür, wie viele Stunden ich programmiert habe.

Zum Glück, sonst wäre die Lizenz vermutlich unbezahlbar geworden.

Er bezahlt dafür, welches Problem gelöst wird und was ihm diese Lösung spart. Damit wurde das Lizenzmodell einfacher. Klare Leistungen, klare Laufzeit und möglichst wenig Verwaltung für den Kunden, und später eine Jahreslizenz, die zu einer Schule oder Organisation deutlich besser passte.

Auch technisch änderte sich dadurch viel. Lizenzbeginn, Aktivierung, Laufzeit, Status, Freischaltungen. Trainingspässe sollten später noch dazukommen. Das Backend wurde damit immer wichtiger, und plötzlich merkte ich: Wenn ein Kunde Geld bezahlt, verändert sich auch meine Verantwortung.

Wenn ich zuhause um Mitternacht irgendeine Funktion zerstöre, ist das mein Problem. Wenn am nächsten Morgen eine Schule damit arbeiten möchte, sieht die Sache anders aus. Damit wurden Tests wichtiger. Sicherungen wichtiger. Updates gefährlicher, und während ich mich immer stärker mit Lizenzen und Kunden beschäftigte, entstand gleichzeitig der nächste große Wunsch.

1Rettungsmittel sollte nicht mehr nur über den Browser oder als Webanwendung erreichbar sein. Ich wollte die App dort sehen, wo Menschen Apps nun einmal suchen. Im App Store.

Kapitel 8. Wie bekommt man das Ding eigentlich in den App Store?

Nachdem ich mit der PWA immer weitergekommen war, entstand irgendwann dieser Gedanke: Ich will 1Rettungsmittel im App Store. Nicht mehr nur über eine Internetadresse. Nicht nur über

den Browser. Sondern als richtige App. Mit Icon. Mit Store-Eintrag. Mit einem Button zum Laden. Das klang gut. Dann kam die erste Erkenntnis. Dafür brauche ich einen Mac.

Bis dahin hatte ich fast alles unter Windows gemacht. Windows kannte ich. Jetzt sollte ich plötzlich in die Apple-Welt. Also schaute ich nach einem Mac und stellte fest: Neu ist das Zeug nicht gerade billig. Meine Lösung war entsprechend bodenständig. Kleinanzeigen. Ein älterer gebrauchter Mac mini. Nicht der Schnellste. Nicht der Neueste. Aber günstig, und er konnte das, was ich brauchte. Zumindest irgendwann. Denn zuerst musste ich lernen: Mac ist nicht Windows.

Ich scrollte mit der Maus. Die Seite bewegte sich für mein Gehirn in die falsche Richtung. Apple nennt das natürlich nicht falsch. Ich nannte es erst mal nervig. Dann die Tastatur. Command statt Control. Andere Fensterlogik. Andere Abläufe. Andere Einstellungen.

Ich wollte eigentlich eine App in den Store bringen. Zunächst lernte ich aber erst mal, einen Mac zu bedienen. Mein gebrauchter Mac mini unterstützte diesen Lernprozess mit viel Ruhe. Sehr viel Ruhe. Xcode öffnen. Warten. Projekt laden. Warten. Build starten. Noch etwas warten.

Man konnte zwischendurch hervorragend über seine Lebensentscheidungen nachdenken. Trotzdem mochte ich das kleine Ding. Denn es tat am Ende genau das, was ich brauchte. Xcode ist Apples Entwicklungsumgebung für Anwendungen auf iPhone und iPad. Meine bestehende Webanwendung musste nun irgendwie in eine native iOS-App kommen.

Dafür nutzte ich Capacitor. Vereinfacht gesagt ermöglicht Capacitor, eine bestehende Webanwendung in eine App für mobile Betriebssysteme einzubetten und dabei auf Funktionen

des Geräts zuzugreifen. Für mich war das ideal. Ich musste nicht bei null anfangen. Theoretisch. Praktisch kamen natürlich neue Begriffe. Bundle Identifier, Signing, Certificates, Provisioning, TestFlight und App Store Connect. Also wieder Schritt für Schritt. Projekt einrichten. iOS hinzufügen. Dateien übertragen. Synchronisieren. Xcode öffnen. Build. Fehler. Fragen. Ändern. Build. Nächster Fehler.

Irgendwann lief 1Rettungsmittel als echtes iOS-Projekt. Dann kam TestFlight. Apples System, mit dem Apps vor der öffentlichen Veröffentlichung auf echten Geräten getestet werden können. Plötzlich installierte ich meine App nicht mehr nur aus meiner Entwicklungsumgebung. Sie kam über Apples eigenes Testsystem auf das Gerät. Das fühlte sich zum ersten Mal richtig offiziell an. Dann kam Apple-Review, und damit die Erkenntnis: Eine App kann technisch funktionieren und trotzdem nicht in den App Store kommen.

Die erste Ablehnung war entsprechend ernüchternd. Unter anderem musste der Kaufprozess anders beziehungsweise vollständiger umgesetzt werden. Also wieder lesen. Ändern. Testen. Neu einreichen. Warten. Das Warten war ungewohnt. Beim Programmieren kann ich etwas ändern und zwei Minuten später sehen, ob es funktioniert. Beim Review gibt man die App ab, und dann wartet man. Natürlich schaut man nicht ständig nach. Nur morgens. Mittags. Nachmittags. Abends, und zwischendurch.

Irgendwann war es so weit. 1Rettungsmittel wurde freigegeben. Im App Store. Ein Icon, ein Name, ein Download-Button. Eigentlich nichts Spektakuläres. Für mich war es riesig. Denn ich sah nicht nur den Store-Eintrag. Ich sah die klickbare PDF. Die ersten Dateien. Die Fehlermeldungen. GitHub. Netlify. Den

gebrauchten Mac mini. Xcode. TestFlight. Die Ablehnung. Das erneute Einreichen, und jetzt konnte jeder die App laden.

Aus: Ich baue da etwas, war plötzlich geworden: Meine App ist im App Store. Das war ein ziemlich gutes Gefühl. Natürlich dauerte es nicht lange, bis die nächste Frage auftauchte. Denn jetzt, wo die App professioneller wurde, wollten die Nutzer nicht mehr nur meine Szenarien verwenden. Sie wollten ihre eigenen.

Kapitel 9. Jetzt bauen die Kunden ihre eigenen Lagen

Bis dahin hatte ich viele Szenarien selbst erstellt. Ich entschied, welche Lage trainiert wird, welche Informationen angezeigt werden und welche Patienten vorkommen. Das funktionierte. Aber je mehr Menschen das System nutzten, desto klarer wurde: Sie wollten ihre eigene Realität abbilden. Nicht irgendeine Lage. Ihre Lage.

Ein eigenes Gebäude. Eine eigene Straße. Das eigene Gelände. Eine bestimmte Übungssituation. Also kam die Frage: Kann man auch eigene Szenarien erstellen? Da war er wieder. Dieser Moment. Ich hatte vorher natürlich versucht, an alles zu denken. Dann stellt jemand trocken eine Frage und innerlich denke ich: Nicht dein Ernst.

Dann denke ich fünf Minuten darüber nach, und meistens folgt: Mist. Hat er recht. Also musste ein Szenario-Editor her. Am Anfang klang das einfach. Titel eingeben. Einsatzstichwort. Patientenzahl. Speichern. Fertig. Natürlich blieb es nicht dabei.

Wenn jemand seine eigene Lage baut, soll sie auch wirklich zu seiner Umgebung passen. Also kamen eigene Fotos dazu. Der Nutzer kann Bilder von seinen eigenen Bereichen verwenden. Von einem Gebäude. Einer Zufahrt. Einem Hof. Einem Übungsgelände.

Dazu kam der Ersteindruck. Was sehen die Teilnehmer, wenn die Lage beginnt? Welche Informationen bekommen sie sofort? Was sollen sie erst mal selbst erkennen? Der Ausbilder sollte genau das eintragen können, was für seine Übung sinnvoll ist, und dann kam die Dynamik.

Eine Lage muss nicht immer statisch bleiben. Patienten können sich verschlechtern. Eine Situation kann sich verändern. Also entstand die Möglichkeit, dynamische Verschlechterungen einzubauen. Auch Gefahrenlagen konnten berücksichtigt werden. Eine Lage kann zunächst relativ statisch beginnen. Dann verändert sich die Situation. Eine Gefahrenlage entsteht oder wird relevant. Plötzlich muss neu bewertet werden.

Wichtig war mir aber auch, nicht jede Kleinigkeit zwanghaft in den Editor zu pressen.

Ein gutes Beispiel ist die Patientenablage. Wie viele Patienten dort beübt werden, muss ich nicht kompliziert im Editor einstellen. Der Ausbilder entscheidet einfach: Heute möchte ich zehn Patienten. Dann verteilt er zehn entsprechende QR-Patienten. Morgen sollen es 25 sein. Dann werden 25 verwendet. Fertig. Nicht jedes Problem braucht ein zusätzliches Menü.

Der Szenario-Editor wurde dadurch nicht einfach nur größer. Er wurde flexibler. Eigene Fotos. Eigener Ersteindruck. Eigene Informationen. Dynamische Patienten. Gefahrenlagen. Eigene Szenarien. Aber trotzdem genug Freiheit für den Instruktor,

während einer Übung spontan zu entscheiden. Damit passierte etwas Wichtiges.

1Rettungsmittel war nicht mehr hauptsächlich mein Trainingssystem. Es wurde ein Werkzeug, mit dem andere ihre eigene Ausbildung gestalten konnten, und natürlich führte genau das zur nächsten Erkenntnis. Je mehr Organisationen das System nutzten, desto öfter hörte ich einen Satz: Bei uns läuft das aber anders.

Kapitel 10. Jeder macht MANV irgendwie anders

Dieser Satz sollte mich noch eine Weile begleiten: Bei uns läuft das aber anders. Am Anfang hatte ich zwangsläufig meine eigene berufliche Realität im Kopf. Die Abläufe, die ich kannte. Die Begriffe, die für mich normal waren. Die Strukturen, mit denen ich arbeitete, und dann zeigte man 1Rettungsmittel Menschen aus anderen Regionen Deutschlands.

Plötzlich stellte man fest: Selbstverständlich ist hier erstaunlich wenig. Bezeichnungen unterscheiden sich. Strukturen unterscheiden sich. Sichtungssysteme unterscheiden sich. Abläufe unterscheiden sich. Was in meiner Region vollkommen normal war, konnte hundert oder zweihundert Kilometer weiter anders organisiert sein.

Das war fachlich interessant. Für die Software war es erst mal anstrengend. Denn wenn ich bundesweit arbeiten wollte, konnte ich nicht einfach sagen: So macht man das. Also musste 1Rettungsmittel flexibler werden, und genau da entstand eine

wichtige Grundhaltung: Die Ausbildung soll sich nicht an meine Software anpassen. Die Software muss sich möglichst an die Ausbildung anpassen.

Software liebt Standards. Die Realität ist davon nicht immer beeindruckt. Also musste ich anfangen, Dinge weniger starr zu bauen. Bezeichnungen mussten anpassbarer werden. Unterschiedliche Sichtungslogiken mussten berücksichtigt werden. Strukturen durften nicht unnötig fest im Code verankert sein. Das bedeutete wieder Umbau, und natürlich dachte ich dabei mehrfach: Wenn ich das am Anfang gewusst hätte, hätte ich das komplett anders aufgebaut. Ein sehr beliebter Satz während meiner Entwicklung. Das Problem daran: Am Anfang konnte ich es nicht wissen.

Ich musste erst mal Menschen aus anderen Bereichen treffen. Ich musste hören, wie sie arbeiten. Ich musste verstehen, was bei ihnen anders läuft, und genau dadurch wurde die Anwendung besser. Gleichzeitig entstand aber eine neue Gefahr: Zu viele Einstellungen. Denn Flexibilität kann eine Anwendung schnell kompliziert machen. Der Kunde möchte zwar seinen eigenen Ablauf abbilden. Er möchte aber nicht vor jeder Übung durch 47 Einstellungen klicken. Wieder dieselbe Aufgabe: Im Hintergrund flexibel. Vorne einfach.

Das war inzwischen fast die zentrale Herausforderung des gesamten Projekts, und wieder halfen die Nutzer. Sie sagten mir, was sie wirklich brauchten. Ich begann langsam zu akzeptieren, dass diese Situationen keine Niederlagen waren. Sie waren Produktentwicklung. Damit wurde 1Rettungsmittel immer interessanter für Schulen und Organisationen außerhalb meines direkten Umfelds. Aber genau dadurch entstand das nächste praktische Problem.

Wenn mehrere Gruppen gleichzeitig trainieren sollen, braucht man mehrere Geräte, und nicht jede Schule hat mal eben zehn Tablets herumliegen.

Kapitel 11. Schönes System. Aber wer hat zehn Tablets?

Je mehr 1Rettungsmittel genutzt wurde, desto größer wurde ein ganz praktisches Problem: Geräte. In meinem Kopf war die Sache zunächst relativ einfach. Wenn mehrere Gruppen gleichzeitig trainieren, braucht man eben mehrere Tablets. Klingt logisch.

Die Realität sah natürlich anders aus. Nicht jede Rettungsdienstschule besitzt zehn Tablets. Nicht jede Feuerwehr hat irgendwo einen kompletten Gerätesatz liegen, der nur darauf wartet, für eine MANV-Ausbildung eingesetzt zu werden, und ich wollte auch nicht, dass mein System am Ende zwar Vorbereitungszeit spart, die Schule dafür aber erst mal mehrere tausend Euro in neue Hardware investieren muss.

Die Idee war deshalb von Anfang an, dass nicht zwingend die Schule alle Geräte stellen muss. Warum sollten Teilnehmer nicht einfach ihre eigenen Tablets mitbringen können? Viele haben ohnehin ein iPad oder ein anderes Tablet zuhause. Genau diese Geräte sollten sich für eine Übung nutzen lassen, ohne dass dafür jedes einzelne Gerät dauerhaft lizenziert werden muss.

Ein Smartphone war für mich dabei keine wirkliche Alternative. Technisch geht vieles auch auf einem Handy. Aber 1Rettungsmittel sollte übersichtlich bleiben. Dafür ist ein Tablet

einfach deutlich besser geeignet. Also musste eine Lösung her, bei der die Schule oder der Instruktor die Kontrolle behält, die Teilnehmer aber trotzdem ihre eigenen Geräte verwenden können. Daraus entstanden die Trainingspässe.

Der Kunde hat seine Jahreslizenz. Dazu bekommt er Trainingspässe als QR-Codes. Wenn ein Teilnehmer sein eigenes Tablet mitbringt, ist dort die App installiert. Der Instruktor stellt den entsprechenden QR-Code zur Verfügung, das Tablet scannt ihn und wird für 24 Stunden freigeschaltet. Fertig.

Keine komplizierte Registrierung. Kein dauerhaftes Benutzerkonto. Keine zusätzliche vollständige Lizenz für ein Gerät, das vielleicht nur an diesem einen Tag gebraucht wird. Genau das war der Gedanke dahinter. Die Schule behält die Freischaltung in der Hand und kann entscheiden, welche Geräte für eine Übung aktiviert werden. Die Teilnehmer können trotzdem ihre eigenen Tablets nutzen.

Für mich war das eine ziemlich elegante Lösung, weil dadurch nicht erst ein kompletter Gerätepark angeschafft werden musste. Natürlich sah es im Hintergrund wieder etwas anders aus. Welche Lizenz hat wie viele Trainingspässe? Welcher Pass wurde bereits verwendet? Wann beginnt die Freischaltung? Wann endet sie? Ist ein Pass noch gültig? Und natürlich wollte ich wieder an möglichst alles denken. Wie immer.

Die eigentliche Lizenzverwaltung dieser Freischaltungen lag bei mir, nicht beim Instruktor. Der Instruktor musste im besten Fall nur den passenden QR-Code bereithalten und konnte sich auf die Ausbildung konzentrieren. Ein zusätzlicher Punkt war die Frage nach den Szenarien.

Ein Trainingspass übertrug nicht automatisch das Szenario auf das andere Gerät. Das war auch gar nicht seine Aufgabe. Er sollte

nur den Zugriff ermöglichen. Eigene Szenarien konnten separat auf weitere Geräte gebracht werden, zum Beispiel schnell per AirDrop.

Auch das war für mich irgendwann eine wichtige Erkenntnis. Nicht jedes Problem braucht sofort eine eigene komplexe technische Lösung. Wenn AirDrop in wenigen Sekunden funktioniert, warum soll ich dafür zwei Wochen programmieren? Die Trainingspässe waren deshalb für mich mehr als nur eine Lizenzfunktion. Sie waren ein gutes Beispiel dafür, wie sich mein Denken verändert hatte. Nicht möglichst viel bauen. Sondern möglichst sinnvoll bauen. Aber wie so oft löste eine gute Lösung direkt das nächste Problem aus.

Denn wenn mehrere Tablets parallel benutzt werden, stellt sich irgendwann eine ziemlich einfache Frage: Was machen die Teilnehmer eigentlich gerade auf diesen Geräten?

Kapitel 12. Was machen die eigentlich auf ihren Tablets?

Diese Frage kam nicht von mir. Natürlich nicht. Ich dachte irgendwann wieder, ich hätte eine ziemlich gute Lösung gebaut. Mehrere Tablets. Trainingspässe. Eigene Szenarien. QR-Patienten. Alles schön. Dann kam sinngemäß aus der Praxis: Wenn ich alleine ausbilde und mehrere Gruppen mit Tablets arbeiten, woher weiß ich eigentlich, was die da machen? Innerlich dachte ich: Nicht dein Ernst. Dann dachte ich kurz darüber nach. Mist. Hat er recht.

Gerade bei einer Patientenablage passiert viel parallel. Der Ausbilder kann nicht gleichzeitig überall stehen. Also musste eine Auswertung her. Wieder einer dieser Sätze, die einfach klingen. Dann speichere ich eben, was passiert. Aber was genau?

Jeden Klick? Jede geöffnete Seite? Jede Sichtung? Jede Änderung? Jede Übergabe? Jede einzelne Sekunde? Technisch kann man sehr viel speichern. Das bedeutet aber noch lange nicht, dass es sinnvoll ist. Wenn eine Übung hinterher 900 Einträge produziert und der Ausbilder erst mal eine halbe Stunde durch technische Ereignisse scrollen muss, habe ich kein Problem gelöst. Also musste ich wieder die entscheidende Frage stellen: Was ist für die Ausbildung wirklich relevant?

Welche Informationen helfen in einer Nachbesprechung? Welche Entscheidungen möchte ein Ausbilder später nachvollziehen können? Welche Zeitpunkte sind wichtig? Und was interessiert niemanden? Eine gute Auswertung ist keine technische Logdatei. Sie soll dem Ausbilder helfen. Nach der Übung. Bei der Besprechung. Schnell. Übersichtlich. Verständlich. Also wurden relevante Aktionen gespeichert. Sichtungen. Zeitpunkte. Entscheidungen. Abläufe, und daraus entstand nach und nach die Einsatznachbesprechung.

Damit bekam 1Rettungsmittel einen neuen Stellenwert. Die App half nicht mehr nur bei der Durchführung. Sie half auch danach. Die Auswertung lag zunächst lokal auf jedem einzelnen Tablet. Das war aus meiner technischen Philosophie heraus logisch. Ich wollte möglichst offline bleiben. Aber aus Sicht eines Ausbilders war die nächste Forderung völlig nachvollziehbar: Kann ich das nicht zentral sehen? Ja. Natürlich. Irgendwann. Denn das war technisch eine ganz andere Nummer.

Zentrale Überwachung bedeutete: Mehrere Geräte müssen miteinander kommunizieren. Daten müssen übertragen werden. Eine zentrale Ansicht muss aktuell bleiben, und das möglichst ohne den ursprünglichen Offline-Gedanken aufzugeben. Diese Idee blieb erst mal im Hintergrund. Später sollte daraus etwas Größeres werden. 1Control. Aber davor stand noch ein anderes Problem. Wenn mehrere Tablets in derselben Übung verwendet werden, sollten Patienten nicht einfach dauerhaft auf einem einzigen Gerät festhängen. In der Realität werden Patienten schließlich auch übergeben.

Kapitel 13. Mehrere Geräte, aber weiterhin offline

Die Frage war eigentlich ziemlich logisch. Ein Patient wird gesichtet. Später wechselt er in einen anderen Bereich. Informationen werden übergeben. Ein anderer Teilnehmer übernimmt. Wenn ich diesen Ablauf realistisch trainieren wollte, musste ein Patient irgendwie von einem Tablet auf ein anderes kommen.

Die einfachste technische Lösung wäre wahrscheinlich gewesen, alles zentral über einen Server laufen zu lassen. Aber genau das wollte ich damals nicht. Ich wollte weiterhin möglichst unabhängig arbeiten. Keine dauerhafte Verbindung nach außen. Keine Übung, die zusammenbricht, nur weil irgendwo WLAN ausfällt. Also musste wieder eine andere Lösung her, und wie so oft landete ich bei einem alten Bekannten. QR-Codes.

Die Dinger wurden langsam zum Schweizer Taschenmesser von 1Rettungsmittel. Die Idee war: Ein gesichteter Patient liegt auf einem Tablet. Die relevanten Daten werden in einen QR-Code gepackt. Das nächste Tablet scannt diesen Code. Der Patient wird übernommen.

Aus Sicht des Nutzers sollte das möglichst simpel sein. Patient übergeben. QR-Code anzeigen. Scannen. Fertig. Im Hintergrund war es natürlich wieder nicht ganz so entspannt. Welche Daten müssen übertragen werden? Wie viel passt in den QR-Code? Was passiert, wenn etwas doppelt übertragen wird? Was passiert, wenn Daten fehlen? Diese Funktion hat mich Nerven gekostet. Sehr viele. Denn eine Übergabe muss zuverlässig funktionieren. Gleichzeitig durfte die Bedienung nicht aus zehn Schritten bestehen.

Also wieder mein inzwischen bekanntes Prinzip: So viel wie nötig. So wenig wie möglich. Mit der QR-Übergabe blieb der Offline Gedanke erhalten. Die Geräte mussten nicht dauerhaft über einen Server miteinander verbunden sein. Trotzdem konnten Informationen weitergegeben werden. Das passte sehr gut zur Grundidee, und während mehrere Geräte immer stärker miteinander arbeiten konnten, blieb die Frage nach einer zentralen Ansicht trotzdem bestehen.

Das Thema war also nicht verschwunden. Ich hatte es nur erfolgreich vertagt. Ein paar Kapitel später sollte es mich wieder einholen. Vorher kam allerdings eine ganz andere Baustelle. Android.

Kapitel 14. Dann kam Android

Nachdem 1Rettungsmittel im Apple App Store war, hatte ich für einen kurzen Moment dieses angenehme Gefühl: Jetzt habe ich das Prinzip verstanden. Ich hatte Xcode überlebt. TestFlight verstanden. Apple-Review erlebt. Eine Ablehnung bekommen. Nachgebessert. Neu eingereicht, und irgendwann stand meine App tatsächlich im App Store. Ich fühlte mich fast ein bisschen cool.

So nach dem Motto: Wenn ich Apple geschafft habe, kann mich jetzt eigentlich nichts mehr schocken. Dann kam der Google Play Store. Rückblickend war diese Selbstsicherheit süß. Denn natürlich war bei Google wieder alles anders. Neue Begriffe. Neue Menüs. Neue Anforderungen. Neue Formulare. Datenschutzangaben. Nachweise. Store Einträge, und Bilder. Sehr viele Bilder.

In irgendwelchen Größen. Natürlich nicht einfach dieselben Größen, die ich schon hatte. Zwischendurch hatte ich teilweise das Gefühl, mehr Zeit mit Bildgrößen zu verbringen als mit meiner App. Dann kam der Testprozess. Zwölf Tester. Vierzehn Tage. Auch das klang erst mal harmlos.

Man braucht zwölf Menschen. Die installieren die App. Dann läuft das eben vierzehn Tage. Fertig. Natürlich war es nicht fertig.

E-Mail-Adressen. Testerlisten. Einladungen. Links. Regionen. Installationen, und dann die wunderbare Frage: Warum zeigt Google jetzt nur eine bestimmte Zahl an, obwohl ich weiß, dass mehr Leute die App installiert haben? Man beginnt plötzlich, einem Verwaltungsportal Dinge beweisen zu wollen, obwohl das

Verwaltungsportal keinerlei Interesse an der eigenen Argumentation hat. Der Test hatte aber auch einen echten Vorteil.

Android ist eine andere Welt. Andere Hersteller. Andere Kameras. Andere Displays. Andere Versionen. Andere Einstellungen, und damit natürlich neue Fehler. Besonders schön waren Kamera und Scanner. Auf einem Gerät funktionierte alles perfekt. Auf dem nächsten verhielt sich die Kamera anders. Oder die Berechtigung. Oder der Fokus. Oder irgendeine Kombination davon. Besonders lehrreich waren meine Familientester.

Irgendwann kam die Rückmeldung: Die Buttons lassen sich nicht richtig anklicken. Also App öffnen. Testen. Bei mir funktionierten sie. Natürlich. Layout ändern. Abstände ändern. Buttons vergrößern. CSS prüfen. Testen. Wieder Rückmeldung: Geht nicht. Ich verbrachte Stunden damit. Irgendwann kam die eigentliche Ursache ans Licht.

Auf dem Testgerät waren aufgrund einer Sehschwäche Schrift und Darstellung deutlich vergrößert worden. Meine App hatte offenbar nicht damit gerechnet, dass ein echter Nutzer sein Gerät tatsächlich an seine eigenen Bedürfnisse anpasst. Frech.

Durch die vergrößerte Darstellung verschoben sich Elemente. Buttons lagen plötzlich ungünstig, und da war wieder eine wichtige Lektion. Ich hatte für mein Gerät programmiert. Der Nutzer benutzt aber sein Gerät. Mit seinen Einstellungen. Seiner Schriftgröße. Seiner Darstellung. Seinen Bedürfnissen. Also musste die Oberfläche angepasst werden. Heute ist die Geschichte lustig. Damals dachte ich eher: Das hättet ihr mir auch früher sagen können. Aber genau dafür testet man.

Diese Android Phase war noch mal ein riesiger Lernblock. Nicht nur technisch. Auch organisatorisch. Apple hatte mich auf seine Art gequält. Google hatte einfach andere Methoden, und

irgendwann war es geschafft. 1Rettungsmittel war auch im Google Play Store. Damit war ein weiterer großer Schritt getan. iOS. Android. Beides, und ich dachte wieder kurz: Jetzt habe ich wirklich einiges abgedeckt.

Das ist meistens der Moment, kurz bevor jemand mit der nächsten Idee kommt.

Kapitel 15. Eigentlich wollte ich Transport gar nicht bauen

Die nächste Idee kam aus der Praxis. Natürlich, und eigentlich wollte ich sie gar nicht. Transport. Mein Fokus war bis dahin klar. Sichtung. Patientenablage. Ausbildung. Auswertung. Das reichte mir erst mal vollkommen. Transportorganisation war für mich noch mal ein eigener Bereich. Fahrzeuge. Ladezone. Bereitstellungsraum. Zielkliniken. Prioritäten. Zeiten. Funk. Führung, und ehrlich gesagt dachte ich: Das muss jetzt nicht auch noch in die App.

In 1Rettungsmittel konnte bereits eine Zielklinik zugewiesen werden. Für viele Übungen war das völlig ausreichend. Aber dann kamen Fragen. Kann man auch ein Rettungsmittel zuweisen? Wie bildet man Transportprioritäten ab? Was ist mit der Ladezone? Was ist mit dem Bereitstellungsraum? Wann ist das Fahrzeug eigentlich wieder verfügbar? Und jedes Mal dachte ich: Leute. Ich wollte eigentlich eine Trainingsapp für Sichtung und Patientenablage bauen. Aber wie immer gab es ein Problem. Die Fragen waren berechtigt. Denn eine MANV-Lage endet nicht in der Patientenablage.

Patienten müssen weiter. Sie müssen transportiert werden. Fahrzeuge müssen organisiert werden. Kliniken müssen ausgewählt werden. Ressourcen sind begrenzt. Also fing ich vorsichtig an. Wirklich vorsichtig. Rettungsmittel auswählen. Zum Beispiel RTW, Rettungswagen. KTW, Krankentransportwagen. RTH, Rettungshubschrauber. Oder andere Fahrzeuge. Dann Zielklinik. Dann Transportpriorität. Mehr wollte ich erst mal nicht. Das war zumindest der Plan. Aber Software hält sich erstaunlich selten an Pläne.

Denn sobald ein Transportmittel ausgewählt wird, entstehen neue Fragen. Wann ist es überhaupt da? Woher kommt es? Steht es im Bereitstellungsraum? Wie gelangt es in die Ladezone? Wann fährt es wieder zurück? Wie lange bleibt es gebunden? Und wenn eine Klinik 40 Kilometer entfernt liegt, kann das Fahrzeug ja nicht drei Minuten später wieder verfügbar sein. Plötzlich wurde aus einer Zusatzfunktion ein ganzer Prozess, und da wurde mir klar: Wenn ich Transport abbilden will, dann muss ich es fachlich sinnvoll machen.

Nicht nur irgendwie. Nicht als hübschen Button. Nicht als Patient transportiert. Sondern so, dass der Ablauf tatsächlich trainierbar wird. Die technische Frage war: Wie bilde ich diese Dinge so ab, dass sie realistisch sind, aber trotzdem einfach bleiben? Denn genau das blieb mein Anspruch. Vorne übersichtlich. Hinten darf es kompliziert sein.

Der Ausbilder soll nicht gleichzeitig fünf Timer bedienen und eine Excel-Tabelle führen müssen. Die Software soll ihm Arbeit abnehmen. Nicht neue Arbeit erzeugen, und damit war der Grundstein für etwas gelegt, das später deutlich größer werden sollte. 1Transport. Zu diesem Zeitpunkt wusste ich nur: Wenn ich das mache, dann richtig.

Kapitel 16. Wenn schon Transport, dann richtig

Irgendwann war klar: Transport ist nicht einfach nur eine Zusatzfunktion. Es ist ein eigener Ausbildungsbereich. Wenn ich größere MANV-Lagen realistisch trainieren möchte, brauche ich in der Realität sehr viele Ressourcen. Fahrzeuge. Personal. Platz. Zeit. Bereitstellungsraum. Ladezone. Kliniken, und vor allem jemanden, der ständig mitdenkt: Welches Fahrzeug ist wann verfügbar? Wann trifft es ein? Wann fährt es zur Klinik? Wann kommt es zurück? Für eine reale Großübung ist dieser Aufwand völlig nachvollziehbar.

Aber genau wie bei meiner ursprünglichen Idee zu 1Rettungsmittel stellte sich wieder die Frage: Geht das nicht einfacher? Nicht als Ersatz für jede reale Übung. Sondern als Möglichkeit, bestimmte Führungs- und Transportabläufe deutlich häufiger und mit weniger Aufwand trainieren zu können.

Wenn ich Transport digital abbilde, kann ich nicht einfach sagen: Hier sind 20 Fahrzeuge. Alle stehen sofort zur Verfügung. Ein Fahrzeug steht vielleicht in fünf Kilometern Entfernung. Das nächste in zwölf. Ein anderes deutlich weiter weg. Also musste die eigene Alarm- und Ausrückeordnung berücksichtigt werden. Die AAO legt vereinfacht gesagt fest, welche Einsatzmittel bei bestimmten Einsatzlagen alarmiert werden.

Der Nutzer sollte seine eigene Struktur eintragen können. Eigene Fahrzeuge. Eigene Funkkennungen. Eigene Entfernungen. Damit auch eigene Eintreffzeiten. Diese Daten sollten bequem vorher am

Rechner eingerichtet werden können. Also entstand die Möglichkeit, die eigene AAO über meine Homepage vorzubereiten.

Dann kam die Ladezone. Auch sie ist nicht unendlich groß. Man kann nicht beliebig viele Fahrzeuge gleichzeitig hineinziehen. Dabei sollte es verschiedene Trainingsmöglichkeiten geben. Man kann zum Beispiel nur die Ladezone trainieren und Fahrzeuge digital abrufen. Das spart Personal. Es spart Platz, und man kann trotzdem die wesentlichen Entscheidungen üben.

Oder man trainiert größer. Dann gibt es tatsächlich einen Führer Bereitstellungsraum. Die Fahrzeuge befinden sich zunächst dort. Wenn sie in die Ladezone geschickt werden, werden sie übergeben. Natürlich wieder über QR-Codes. Was sonst. Auch die Kliniken mussten realistisch werden.

Der Nutzer kann eigene Zielkliniken hinterlegen. Mit eigenen Entfernungen. Wenn ein Fahrzeug einen Patienten 40 Kilometer weit transportiert, ist es entsprechend lange gebunden. Genau solche Dinge waren mir wichtig. Denn wenn man Führung trainiert, muss Ressourcenknappheit spürbar sein.

Die App übernimmt dabei möglichst viel automatisch. Eintreffzeiten. Fahrzeugbindung. Verfügbarkeit. Transportzeiten. Der Ausbilder soll nicht fünf Timer gleichzeitig bedienen. Er soll beobachten, steuern, nachfragen und auswerten. Also wieder derselbe Grundgedanke: Technik soll Arbeit abnehmen. Nicht neue Arbeit produzieren, und damit kommt wieder der wirtschaftliche Aspekt ins Spiel.

Eine große Übung braucht normalerweise viel Personal und Material. Wenn ich bestimmte Teile davon digital abbilden kann, spare ich nicht nur Zeit. Ich spare reale Ressourcen. Ich wollte

keinen billigen Zusatz. Kein Häkchen bei: Transport können wir jetzt auch. Wenn ich diesen Bereich baue, dann fachlich sinnvoll.

Irgendwann musste ich mir deshalb eine neue Frage stellen: Soll das überhaupt noch alles dieselbe App sein?

Kapitel 17. Aus einer Funktion wird eine eigene App

In 1Rettungsmittel konnte ein Patient bereits aus der Patientenablage einer Zielklinik zugewiesen werden. Für viele Übungen war das völlig ausreichend. Aber 1Transport sollte etwas anderes werden. Nicht nur eine zusätzliche Transportfunktion. Sondern eine eigene Trainingsumgebung für Transportorganisation.

Bereitstellungsraum. Ladezone. Funk, Aufgabenbereiche, Fahrzeuge, eigene AAO, eigene Kliniken, Eintreffzeiten, Transportzeiten und Fahrzeugbindung, und damit wurde mir irgendwann klar: Das wird zu groß für einen weiteren Menüpunkt. Ich wollte 1Rettungsmittel nicht mit Funktionen überladen, bis irgendwann niemand mehr versteht, wo eigentlich was passiert. Also entstand 1Transport als eigenständige App. Nicht als Konkurrenz zu 1Rettungsmittel. Sondern als Ergänzung.

1Rettungsmittel konzentriert sich auf die Ausbildung rund um Sichtung, Patientenablage, Patientenentwicklung und weitere Abläufe. 1Transport nimmt sich gezielt die Transportorganisation vor. Das war für mich ein wichtiger Schritt. Denn bis dahin war mein Reflex häufig: Noch eine Funktion. Noch eine Möglichkeit.

Noch etwas integrieren. Bei 1Transport entschied ich bewusst: Das bekommt einen eigenen Platz.

1Transport sollte dabei bewusst niedrigschwellig bleiben. Bis zu 20 Patienten kann die App kostenlos genutzt werden. Der Nutzer kann sie herunterladen, seine Struktur vorbereiten, trainieren, ausprobieren und selbst entscheiden, ob das System einen Mehrwert bringt.

Zum Start war die App zunächst im Apple App Store verfügbar. Android sollte später folgen. Auch da hatte ich inzwischen zumindest eine Sache gelernt: Nicht jede Baustelle muss gleichzeitig geöffnet werden. Zumindest versuchte ich, mich daran zu halten.

Mit 1Transport entstand aus dem ursprünglichen Projekt langsam etwas Größeres. 1Rettungsmittel. 1Transport, und im Hintergrund immer mehr gemeinsame Gedanken zu Ausbildung, Organisation und digitalen Prozessen. Was im September 2024 mit einer klickbaren PDF begonnen hatte, entwickelte sich langsam zu einem kleinen System. Eigentlich ein ziemlich guter Moment, um sich kurz zurückzulehnen. Leider hatte ich dafür keine Zeit. Denn parallel musste ja noch jemand dafür sorgen, dass überhaupt irgendwer davon erfährt.

Kapitel 18. Die App baut sich nicht allein bekannt

Eine der ernüchterndsten Erkenntnisse war: Eine gute App verkauft sich nicht automatisch. Sie steht nicht morgens auf. Schreibt keine E-Mails. Macht keine Instagram-Beiträge. Fährt

nicht zu Kunden, und sie erklärt auch niemandem von selbst, warum sie ein Problem löst. Das musste ich offenbar alles selbst machen. Also kam neben Softwareentwicklung noch ein komplett neues Themenfeld dazu. Marketing, Homepage, Instagram, YouTube, Presse und Werbung.

Alles Dinge, bei denen mein Wissensstand anfangs ungefähr auf dem Niveau meiner ersten Programmierkenntnisse lag. Instagram war dabei eine besondere Erfahrung. Ich wusste, dass Social Media wichtig sein kann. Reels, Storys, Beiträge, kurze Videos, Reichweite. Für andere völlig selbstverständlich. Für mich nicht unbedingt. Zum Glück hatte ich Kevin.

Ein Kollege, der mir am Anfang viel gezeigt hat. Wie man Beiträge aufbaut. Wie Videos funktionieren. Was man überhaupt zeigen kann. Das war eine echte Hilfe. Denn allein hätte ich vermutlich erst mal versucht, ein Reel wie eine Dienstanweisung aufzubauen.

Ob ich Instagram heute vollständig verstanden habe? Nein. Ganz klar nein, und meine Lieblingsbeschäftigung ist es bis heute nicht. Ich entwickle deutlich lieber eine neue Funktion, als zwanzig Minuten darüber nachzudenken, welcher Text unter ein Video gehört. Aber es gehört dazu. Also machte ich Videos. Nahm Dinge auf. Schnitt. Löschte. Nahm neu auf, und natürlich änderte sich währenddessen die App. Damit war das Video plötzlich veraltet. Also noch mal. Besonders schön war das bei YouTube.

Ich hatte relativ früh angefangen, Erklärvideos zu machen. Das war sinnvoll. Ein Nutzer kann sich ansehen, wie eine Funktion funktioniert. Praktisch sah es oft so aus: Video aufnehmen, Funktion erklären, hochladen. Ein paar Wochen später änderte ich die App: Button woanders, Ablauf verbessert, Menü angepasst, Video wieder alt. Also neu.

Wenn man sehr viel Zeit verbrauchen möchte, ohne eine einzige neue Funktion zu entwickeln, kann ich Erklärvideos für eine sich ständig verändernde App empfehlen. Dann war da die Homepage.

Auch sie musste erst mal entstehen. Was muss drauf? Was interessiert den Nutzer? Wie erkläre ich 1Rettungsmittel, ohne einen Roman auf die Startseite zu schreiben? Wo kommen Preise hin? Wie integriere ich QR-Codes zum Download? Was ist gerade der aktuelle Funktionsstand?

Dann kam 1Transport. Homepage ändern. Android kam dazu. Homepage ändern. Neue Funktion. Homepage ändern. Neue Lizenz. Homepage ändern. Auch eine Webseite ist nie einfach fertig. Parallel kamen Presse und Werbung. Zeitungen anschreiben, Projekt vorstellen, Kontakte suchen und hoffen, dass jemand das Thema interessant findet.

Manchmal kam eine Antwort. Manchmal nicht. Manchmal ein gutes Gespräch. Manchmal überhaupt nichts, und dann merkt man irgendwann: Werbung kostet Geld. Teilweise sehr viel Geld, und nur weil etwas Geld kostet, heißt das noch lange nicht, dass daraus ein Kunde entsteht. Das war für mich eine andere Art von Unsicherheit.

Bei Code gibt es am Ende meistens zwei Zustände. Funktioniert oder funktioniert nicht. Marketing kennt deutlich mehr Grautöne, und trotzdem meldete sich immer wieder jemand. Eine Schule. Ein Ausbilder. Ein Tester. Jemand mit einer guten Idee.

Nicht jeder wurde Kunde. Aber fast jeder gute Kontakt brachte etwas. Feedback. Wissen. Motivation. Kontakte. Oder einfach den nächsten Gedanken, und gerade in einem so spezialisierten Bereich kann genau das enorm wichtig sein. Während all das lief, musste ich aber noch eine weitere Rolle übernehmen. Denn je mehr Nutzer und Lizenzen dazukamen, desto häufiger passierte

etwas, das irgendwann jedes digitale Produkt betrifft: Jemand hat ein Problem.

Kapitel 19. Plötzlich bin ich auch noch Admin

Eine App zu entwickeln ist eine Sache. Eine App zu betreiben ist etwas anderes. Das habe ich irgendwann ziemlich deutlich gelernt. Denn plötzlich gab es Kunden. Lizenzen. Freischaltungen. Trainingspässe. Fragen. Fehler, und natürlich treten Probleme nicht nur dann auf, wenn ich gemütlich zuhause am Rechner sitze.

Wenn irgendwo eine Lizenz nicht sauber aktiviert wurde, wollte ich nicht erst Stunden später reagieren können. Wenn jemand am nächsten Morgen eine Ausbildung durchführen wollte, half es wenig zu sagen: Ich schaue heute Abend mal. Also entstanden eigene Admin-Bereiche. Nicht für den Nutzer. Für mich.

Übersichten, Lizenzstatus, Aktivierungen, Freischaltungen und die Möglichkeit, schnell nachzusehen, was gerade passiert, und natürlich mussten diese Zugriffe mobil funktionieren. Denn ein Admin-System, das nur zuhause am großen Rechner erreichbar ist, hilft wenig, wenn mich unterwegs jemand anruft. Auch hier galt wieder: Im besten Fall merkt der Nutzer davon nichts.

Wenn alles läuft, interessiert niemanden, wie die Lizenz technisch geprüft wird. Wenn etwas nicht läuft, möchte der Nutzer vor allem eins: Dass es schnell wieder funktioniert. Support wurde damit Teil des Produktes. Parallel musste ich mich intensiv mit Datenschutz beschäftigen.

Die entscheidenden Fragen waren für mich: Was muss ich speichern? Was darf ich speichern? Und was will ich überhaupt speichern? Denn nur weil ich technisch Daten speichern kann, heißt das nicht, dass ich es tun sollte. Mein Grundsatz wurde deshalb: So wenig wie möglich.

Wenn eine Information für den Betrieb nicht gebraucht wird, brauche ich sie nicht. Das macht vieles einfacher und sicherer. Natürlich musste trotzdem einiges verwaltet werden. Lizenzen, Laufzeiten, Aktivierungen, Trainingspässe und Kundenstatus, und irgendwann reicht dafür keine Erinnerung mehr. Am Anfang denkt man noch: Das weiß ich doch. Später fragt man sich: War das jetzt der Kunde mit dem Testzugang? Oder der mit der Jahreslizenz? Wann wurde die aktiviert? Wie viele Trainingspässe waren da noch mal drin? Spätestens dann braucht man ein System.

Damit entstand im Hintergrund immer mehr Infrastruktur. Nicht besonders spektakulär. Aber notwendig. Backups, Server, Adminzugänge, Datenbankstrukturen, Lizenzprüfungen, Fehleranalyse und Support. All diese Dinge sieht man in keinem Werbevideo. Sie machen aber den Unterschied zwischen einem Projekt und einem Produkt. Denn ein Projekt darf auch mal nur auf meinem Rechner funktionieren. Ein Produkt muss morgen noch funktionieren, und möglichst auch nächste Woche.

Auch hier half mir wieder eine Eigenschaft, die mich durch das gesamte Projekt begleitet hat. Ich wollte Probleme verstehen. Nicht einfach wegdrücken. Das hatte ich schon als Kind erlebt. Mein Vater war genauso. Wenn etwas kaputtging, war die erste Frage selten: Was kostet neu?

Sondern: Warum ist es kaputt? Kann man das reparieren? Was ist die Ursache? Das habe ich ziemlich stark übernommen. Bei einem

Thema sollte ich damit allerdings zunächst scheitern. Richtig scheitern. Das Thema hieß: Zentrale Überwachung.

Kapitel 20. Ich hatte es schon einmal versucht und aufgegeben

Die Idee, mehrere Geräte miteinander zu verbinden, war nicht neu. Eigentlich hatte ich es schon deutlich früher versucht. Damals dachte ich: Wenn mehrere Tablets in einer Übung eingesetzt werden, müsste es doch möglich sein, die Daten irgendwie zusammenzubringen. Vielleicht über einen kleinen lokalen Server. Klingt erst mal logisch. Also machte ich das, was ich inzwischen ziemlich oft gemacht hatte. Ich kaufte Technik. Kleine Server. Hardware. Probierte Dinge aus. Las mich ein. Fragte KI. Programmierte. Testete. Änderte. Noch mal, und noch mal. Sehr viel Zeit später war das Ergebnis ernüchternd.

Es funktionierte nicht. Zumindest nicht so, dass ich es ernsthaft einem Kunden hätte geben können. Ich hatte unglaublich viel Zeit investiert, und irgendwann war ich einfach nur noch genervt. Also sagte ich mir: Dann gibt es das eben nicht. Fertig.

Das Problem war nur: Die Kunden akzeptierten diese Entscheidung nicht unbedingt. Immer wieder kam die Frage: Kann ich eigentlich zentral sehen, was auf den anderen Tablets passiert? Oder: Kann ich live verfolgen, was während der Übung gemacht wird? Und jedes Mal dachte ich: Ja. Wäre schön. Aber ihr habt keine Ahnung, was ihr da gerade von mir verlangt.

Ein Kunde sieht eine App. Er sieht Funktionen. Er sieht, was schon möglich ist. Dann fragt er: Kann das auch noch das? Für ihn ist

das eine völlig normale Frage. Was er nicht sieht: Ich habe zwar inzwischen ziemlich viel gelernt. Aber ich bin immer noch kein ausgebildeter Serverentwickler. Der Kunde denkt vielleicht: Warum baut er das nicht einfach?

Ich denke: Weil ich erst mal herausfinden muss, wie das überhaupt geht. Aber genau da kam wieder mein Ehrgeiz ins Spiel. Der kommt nicht von ungefähr. Mein Vater hat da einen großen Anteil. Problem erkennen. Ursache suchen. Reparieren. Nicht sofort wegwerfen. Nicht sofort neu kaufen. Erstmal verstehen.

Wobei wir uns bei einem Thema nie vollständig einig sind: bei der Wasserwaage. Für mich ist eine Wasserwaage relativ einfach. Zwei Striche. Eine Blase. Blase zwischen den Strichen. Gerade. Fertig. Für meinen Vater war das offenbar nur die grobe Vorstufe von gerade. Die Blase musste möglichst so exakt sitzen, dass man den Eindruck bekam, die Blase müsste links und rechts genau an den Strichen ausgerichtet werden. Damals fand ich das teilweise etwas übertrieben.

Heute erwische ich mich dabei, wie ich an einer Funktion zum zehnten Mal etwas ändere, obwohl sie eigentlich schon funktioniert. Offenbar vererbt sich Perfektionismus über Wasserwaagen. Beim Thema Servertechnik war irgendwann der Punkt erreicht, an dem ich Abstand brauchte, und genau dieser Abstand brachte irgendwann die entscheidende Idee.

Nicht zuhause. Nicht nachts am Rechner. Sondern während eines Staatsexamens für Notfallsanitäter. Ich saß dort als Prüfer. In einer Pause kam plötzlich dieser Gedanke: Oh Mann. Bist du eigentlich blöd? Es gibt solche Prinzipien doch längst. Systeme, bei denen man auf einem Gerät etwas einstellt und es auf einem anderen angezeigt wird. Das technische Grundprinzip existiert.

Warum versuche ich eigentlich, alles von null neu zu erfinden? Also begann ich noch mal zu recherchieren.

Wie machen andere Systeme das? Wie können Geräte lokal miteinander kommunizieren? Was brauche ich wirklich? Und was brauche ich eben nicht? Plötzlich sah das Problem kleiner aus. Ich brauchte keinen riesigen Server. Ich musste genau das lösen, was mein Produkt braucht. Geräte verbinden. Daten übertragen. Live anzeigen. Mehr nicht.

Dazu kam etwas Entscheidendes. Die KI hatte sich inzwischen weiterentwickelt, und ich ebenfalls. Ich nutzte mittlerweile leistungsfähigere Werkzeuge. Ich bezahlte inzwischen auch für bessere KI-Unterstützung. Die Systeme konnten größere Teile meines Projekts verstehen. Zusammenhänge besser erfassen. Code gezielter verändern. Fehler besser einordnen.

Was früher wie ein riesiger Berg wirkte, ließ sich plötzlich in kleinere Aufgaben zerlegen. Verbindung herstellen, Geräte erkennen, Daten senden, Daten empfangen, Anzeige aktualisieren, testen, Fehler beheben, und irgendwann passierte etwas, womit ich bei meinem ersten Versuch nicht gerechnet hätte. Es funktionierte.

Mehrere Geräte konnten miteinander kommunizieren, und auf einem zentralen Gerät konnte ich sehen, was auf den anderen passierte. Live. Das war der Beginn von 1Control. 1Control soll die bestehende Jahreslizenz erweitern. Ein Instruktor soll eine Übung erstellen und während der Durchführung zentral verfolgen können, was auf den beteiligten Geräten passiert.

Zum Zeitpunkt, an dem ich diese Geschichte schreibe, steht 1Control kurz vor diesem Punkt. Ich teste viel. Vieles läuft bereits. Fehler tauchen auf. Fehler werden behoben, und wahrscheinlich wird irgendwann jemand bei einer Vorstellung ganz trocken

fragen: Kann das eigentlich auch noch …? Dann werde ich innerlich wieder denken: Nicht dein Ernst, und vermutlich wird er wieder recht haben.

Vor zwei Jahren hatte ich eine klickbare PDF. Heute sehe ich mehrere Tablets live auf einem zentralen Gerät. Wenn mir das damals jemand erzählt hätte, hätte ich wahrscheinlich gesagt: Klingt cool. Keine Ahnung, wie das gehen soll. Heute weiß ich es zumindest ungefähr, und manchmal reicht genau das.

Kapitel 21. Was ich komplett unterschätzt habe

Wenn man am Anfang eine App bauen will, denkt man vor allem an eins: Die App. Klingt logisch. Man denkt an Funktionen. An Oberfläche. An Technik. Vielleicht noch an Design. Was man deutlich weniger im Kopf hat, ist alles, was später daneben entsteht.

Homepage, Datenschutz, Support, Lizenzen, Admin-Bereiche, Stores, Werbung, Erklärvideos, Presse, Kundenkommunikation, Updates, Backups, Bilder, Texte, und ungefähr hundert andere Dinge, von denen man am Anfang noch gar nicht weiß, dass man sie irgendwann dringend braucht. Genau das habe ich komplett unterschätzt. Nicht ein bisschen. Komplett.

Ich dachte lange: Wenn die App erst mal funktioniert, dann ist der größte Teil geschafft. Heute weiß ich: Dann fängt ein anderer großer Teil erst an.

Eine App im App Store bringt nichts, wenn niemand weiß, dass sie existiert. Eine Jahreslizenz bringt nichts, wenn ich sie nicht

vernünftig verwalten kann. Ein Kunde bringt mir wenig, wenn ich bei einem Problem erst abends zuhause an meinen Rechner muss. Diese Themen liefen irgendwann alle gleichzeitig.

Ich programmierte. Dann änderte ich die Homepage. Dann musste ein Kunde freigeschaltet werden. Dann kam eine Frage zum Datenschutz. Dann wollte jemand ein Erklärvideo. Dann fiel mir auf, dass mein letztes YouTube-Video schon wieder veraltet war. Also neu aufnehmen. Dann Instagram. Dann wieder Code.

Irgendwann bestand das Projekt längst nicht mehr nur aus Programmierung. Es war eher ein kleiner Betrieb. Nur ohne Personal. Besonders viel Zeit fraß alles, was mit Erklären zu tun hatte. YouTube war dafür ein gutes Beispiel. Die Idee war sinnvoll. Funktionen erklären. Damit Kunden nicht für jede Kleinigkeit fragen müssen. Dann änderte ich die App. Natürlich.

Ein Button war plötzlich woanders. Eine Funktion sah anders aus. Ein Menü war neu, und das Video war praktisch schon wieder alt. Instagram war ähnlich. Ich wusste, dass Social Media dazugehört. Ich hatte aber keinen Plan, wie man vernünftig Reels macht.

Kevin hat mir am Anfang wirklich geholfen. Trotzdem ist Instagram bis heute nicht meine Lieblingsbeschäftigung. Ich kann inzwischen deutlich besser mit Code umgehen als mit Reels. Das hätte ich 2024 wahrscheinlich auch nicht erwartet. Aber es gehört dazu.

Das Gleiche galt für Presse und Werbung. Zeitungen anschreiben. Kontakte suchen. Produkt erklären. Hoffen, dass jemand Interesse hat. Manchmal kam etwas zurück. Manchmal nicht, und natürlich stellte ich irgendwann fest: Werbung kann ziemlich teuer sein.

Vor allem dann, wenn man noch nicht genau weiß, was überhaupt funktioniert. Bei Code ist es meistens relativ eindeutig.

Funktioniert. Oder funktioniert nicht. Bei Werbung kann etwas gut aussehen, Reichweite bringen und trotzdem keinen einzigen Abschluss. Diese Unsicherheit musste ich lernen auszuhalten.

Ein weiterer Punkt, den ich massiv unterschätzt hatte, war mein eigener Hang dazu, schon im Voraus an alles denken zu wollen.

Bei jeder neuen Funktion begann in meinem Kopf sofort eine Liste. Was passiert, wenn jemand zweimal klickt? Was passiert ohne Internet? Was passiert mit zehn Geräten gleichzeitig? Was passiert, wenn eine Lizenz genau während einer Übung abläuft?

Manchmal war diese Gründlichkeit gut. Sie verhinderte Fehler. Sie machte Funktionen stabiler. Aber oft war sie einfach nur teuer. Nicht unbedingt in Geld. In Zeit, und in Nerven. Ich programmierte Lösungen für Probleme, die noch gar nicht existierten, und dann kam regelmäßig ein Kunde. Ich stellte das System vor. War überzeugt, diesmal wirklich an alles gedacht zu haben, und dann sagte jemand ganz trocken: Geht das eigentlich auch? Innerlich dachte ich: Nicht dein Ernst. Das Problem war nur: Sehr oft hatte der Kunde recht. Rückblickend war das eine der wichtigsten Erkenntnisse überhaupt.

Man kann versuchen, an alles zu denken. Aber die besten Fragen kommen oft von Menschen, die das System zum ersten Mal sehen. Genau das habe ich am Anfang komplett unterschätzt.

Kapitel 22. Was ich heute anders machen würde

Wenn ich heute noch mal im September 2024 anfangen würde, würde ich vieles genauso machen. Ich würde wieder anfangen.

Auch ohne perfekten Plan. Auch ohne vollständige technische Kenntnisse. Auch mit einer ziemlich einfachen ersten Lösung. Wahrscheinlich wieder mit etwas, das andere belächeln würden. Denn genau das war rückblickend richtig. Die klickbare PDF war technisch nichts Besonderes. Aber sie war der Start. Was ich anders machen würde, beginnt erst danach.

Ich würde früher mit echten Nutzern sprechen. Viel früher. Nicht erst dann, wenn ich glaube, dass eine Funktion fertig ist. Denn ich habe unglaublich viel Zeit damit verbracht, Lösungen für Probleme zu bauen, die nur in meinem Kopf existierten.

Das erste Lizenzmodell war dafür ein perfektes Beispiel. Drei Stunden, zwölf Stunden, vierundzwanzig Stunden, sieben Tage, dreißig Tage: alles über PayPal, alles automatisch. Technisch fand ich das richtig gut, und irgendwann stellte ich fest: Rettungsdienstschulen wollen das so gar nicht. Ich hatte die falsche Frage beantwortet.

Ich hatte gefragt: Wie kann ich möglichst viele flexible Laufzeiten technisch sauber abbilden? Die bessere Frage wäre gewesen: Wie möchte eine Rettungsdienstschule das überhaupt kaufen? Heute würde ich diese zweite Frage zuerst stellen. Ich würde auch nicht mehr versuchen, von Anfang an jeden theoretisch möglichen Sonderfall zu lösen.

Für zehn reale Sonderfälle hatte ich vorher wahrscheinlich dreißig theoretische abgesichert, die nie jemand brauchte. Das kostete Zeit. Sehr viel Zeit. Heute würde ich häufiger sagen: Bau die Grundfunktion. Teste sie. Schau, was wirklich passiert. Dann verbessere sie. Nicht jedes zukünftige Problem braucht heute schon eine Lösung.

Ich würde außerdem früher akzeptieren, dass gut genug zum Testen nicht dasselbe ist wie schlecht. Ich wollte Dinge oft zu

sauber machen, bevor ich sie überhaupt jemandem gezeigt habe. Das steckt vermutlich tief in mir.

Dieser Hang zur Genauigkeit hat mir bei 1Rettungsmittel enorm geholfen, aber er hat mich auch sehr viel Zeit gekostet. Heute würde ich deshalb stärker unterscheiden: Muss das perfekt sein? Oder muss es erst einmal funktionieren und getestet werden? Das ist nicht dasselbe.

Ich würde technische Änderungen auch weniger spontan live stellen.

Gerade am Anfang war die Versuchung riesig. Neue Funktion gebaut. Dreimal getestet. Sieht gut aus. Online damit. Was soll schon passieren? Eine Menge. Ich würde außerdem viel früher sauber dokumentieren, warum ich bestimmte Dinge gebaut habe. Nicht nur wie.

Ein paar Monate später schaut man auf Code und versteht vielleicht noch, was er macht. Aber man weiß nicht mehr unbedingt, warum man sich damals für genau diesen Weg entschieden hat, und dann sitzt man vor seinem eigenen Projekt und denkt: Welcher Idiot hat das denn so gebaut? Dann erinnert man sich. Ach ja. Ich.

Ich würde auch früher verstehen, dass Marketing keine Aufgabe für später ist. Ich dachte lange: Erst die App richtig gut machen. Dann kümmern wir uns um den Rest. Nur kommt dieser Punkt nie. Ich würde auch früher verstehen, dass nicht jedes gute Gespräch zu einem Auftrag führen muss, um wertvoll zu sein.

Vielleicht kam eine Idee aus dem Gespräch. Vielleicht hat jemand ein Problem genannt, das ich später gelöst habe. Vielleicht kam dadurch ein anderer Kontakt. Das ist ebenfalls wertvoll. Ich würde vermutlich auch früher lernen, bei manchen Dingen

Abstand zu nehmen. Nicht aufzugeben. Abstand. Das ist etwas anderes. Beim Thema 1Control war genau der Abstand entscheidend.

Mein erster Versuch mit kleiner Servertechnik hatte unglaublich viel Zeit verschlungen. Es funktionierte nicht. Ich war genervt. Ich habe das Thema irgendwann begraben, und später, in einer völlig anderen Situation, kam die Idee zurück. Heute würde ich deshalb öfter sagen: Wenn du seit drei Stunden dieselbe Fehlermeldung anstarrst, wird sie durch die vierte Stunde wahrscheinlich nicht hübscher. Geh weg. Mach etwas anderes. Denk später noch mal darüber nach.

Ich würde außerdem früher akzeptieren, dass KI nicht automatisch bedeutet, dass man kein eigenes Verständnis mehr braucht. Ganz im Gegenteil. Je leistungsfähiger die Werkzeuge wurden, desto wichtiger wurde es für mich zu verstehen, was ich eigentlich möchte.

Am Anfang war ich froh, wenn mir KI irgendeinen Code gab, der funktionierte. Heute kann ich viel genauer formulieren, was passieren soll. Ich kann Fehler besser einordnen. Ich erkenne schneller, wenn ein Vorschlag Quatsch ist. Aber die wichtigste Entwicklung war vielleicht gar nicht die KI. Sondern meine Fähigkeit, bessere Fragen zu stellen. Nicht fragen: Bau mir eine App.

Sondern: Welches Problem möchte ich lösen? Was soll der Nutzer genau tun? Was muss dabei passieren? Was darf nicht passieren? Was ist jetzt wirklich notwendig? Und trotz all dieser Dinge, die ich heute anders machen würde, gibt es einen Punkt, den ich genauso wieder machen würde. Ich würde anfangen, bevor ich bereit bin.

Wenn ich im September 2024 gewartet hätte, bis ich genug über Programmierung, Server, App Stores, Datenschutz, Lizenzmodelle, Marketing und Vertrieb weiß, würde es 1Rettungsmittel wahrscheinlich heute nicht geben. Ich habe nicht erst alles gelernt und danach angefangen. Ich habe angefangen, und musste deshalb lernen. Also ja. Ich würde heute vieles anders machen. Aber eines würde ich auf keinen Fall ändern: Ich würde die klickbare PDF wieder bauen.

Kapitel 23. Ohne die Menschen wäre es nur Code geblieben

Wenn ich heute auf die letzten zwei Jahre zurückblicke, denke ich natürlich an Technik: an Code, Fehlermeldungen, App Stores, Server, QR-Codes und Nächte vor dem Rechner. Aber wenn ich ehrlich bin, ist das nur ein Teil der Geschichte.

Der wichtigere Teil sind die Menschen. Denn 1Rettungsmittel ist nicht in einem stillen Raum entstanden, sondern durch Fragen, Rückmeldungen, Tests, Zweifel, Ermutigung und manchmal auch durch Sätze, die ich im ersten Moment gar nicht hören wollte.

Ganz am Anfang war da Carina. Sie hat nicht einfach nur einmal gesagt: Mach mal. Sie hat die Idee über die Jahre mitgetragen. Sie hat zugehört, wenn aus einer kleinen Funktion plötzlich wieder ein ganzer Abend wurde. Sie hat getestet, mitgedacht und wahrscheinlich öfter Geduld bewiesen, als man vernünftigerweise erwarten darf.

Und sie musste sich natürlich immer wieder neue Versionen ansehen. Neue Funktion. Neues Problem. Neue Idee. Wieder eine

Änderung. Dann doch wieder anders. Irgendwann war sie vermutlich eine der meistgetesteten Nichtrettungsdienstlerinnen Deutschlands.

Als Friseurmeisterin hat sie heute wahrscheinlich einen besseren Blick dafür, wann ein Patient ein Tourniquet braucht, als manche Menschen, die beruflich näher an dem Thema dran sind. Ob sie das jemals wollte? Wahrscheinlich nicht. Aber sie kann es jetzt.

Auch als Patientin musste sie früh herhalten: Fotos, Verletzungsmuster, Schminken, Darstellen, später sogar in der Garage. Und natürlich blieb es nicht bei ihr. Schwiegereltern, Freunde und Menschen aus meinem Umfeld mussten irgendwann Verletzte spielen, damit meine App besser wurde. Wenn man es so schreibt, klingt es etwas merkwürdig. War es wahrscheinlich auch.

Aber niemand sagte: Lass den Quatsch. Sie machten mit.

Auch meine Mutter gehörte von Anfang an zu den Menschen, die mich eher bestärkt als gebremst haben. Wenn ich ihr erzählte, dass ich gerade eine App baue, kam nicht: Was soll der Quatsch? Sondern eher: Probier es. Mach weiter. Schau, was daraus wird.

Natürlich könnte man sagen: Klar, ist ja die Mutter. Die muss ihren Jungen unterstützen. Wer meine Mutter kennt, weiß aber: Nein, muss sie nicht. Wenn ihr etwas nicht passt, sagt sie es. Direkt. Ins Gesicht. Und ehrlich gesagt glaube ich, dass ich auch davon etwas mitgenommen habe.

Mein Vater hat meinen Ehrgeiz und meine Art, Probleme anzugehen, stark geprägt. Vieles davon habe ich wahrscheinlich übernommen, ohne es lange bewusst zu merken.

Es gab Nicole, Kollegin und gute Freundin, die mir Tipps gab, mitdachte und half. Dann Kevin, der mir bei Instagram half,

obwohl ich wahrscheinlich bis heute nicht vollständig verstanden habe, warum manche Reels funktionieren und andere nicht.

Und es gab Sven von Atem-Life. Am Anfang durchaus skeptisch, später immer interessierter, und über die Zeit ein ehrlicher Ratgeber, wenn ich mal wieder mit einer Idee, einer Frage oder einem halbfertigen Gedanken ankam.

Dazu kamen Kolleginnen und Kollegen bei der Berufsfeuerwehr, die Fehler fanden, Tipps gaben, Dinge hinterfragten und trotzdem Rückendeckung gaben. Das NAZ Goslar testete früh, dachte mit und wurde später mein erster Kunde.

Schulen, Organisationen, Tester, Interessenten, Kunden und auch Menschen, die nicht gekauft haben, haben trotzdem etwas beigetragen. Nicht jeder Kontakt brachte Geld. Aber erstaunlich viele brachten etwas anderes: Wissen, Feedback, Motivation, Kontakte oder eine neue Perspektive.

Das klingt im ersten Moment vielleicht wie eine Ansammlung kleiner Dinge. Ist es aber nicht. Wenn man ein Projekt über so lange Zeit durchzieht, helfen nicht nur die großen Momente. Es sind oft die kleinen: ein Foto, eine Rückmeldung, ein ehrlicher Satz, ein Test, jemand, der sich fünf Minuten Zeit nimmt.

Auch Kritik gehört dazu. Nicht jeder muss die eigene Idee verstehen, daran glauben oder sie gut finden. Entscheidend ist nur: Glaube ich selbst noch daran? Und gibt es Menschen, die das Problem ebenfalls sehen?

Die Antwort darauf war über die gesamte Entwicklung immer wieder: Ja.

Ich habe in den letzten Jahren viele Menschen kennengelernt, die Ausbildung besser machen wollen. Im Rettungsdienst, bei Feuerwehren, bei Hilfsorganisationen, in Schulen und im

Ehrenamt. Menschen, die nach ihrer eigentlichen Arbeit noch Übungen vorbereiten, Material organisieren und anderen etwas beibringen.

Warum? Weil sie wollen, dass Einsatzkräfte im Ernstfall besser vorbereitet sind. Am Ende geht es genau darum.

Nicht um eine App, nicht um einen QR-Code und nicht um irgendeine technische Funktion. Es geht darum, dass Einsatzkräfte im Einsatz besser handeln können, und dadurch vielleicht irgendwann jemandem besser geholfen wird.

Natürlich ist 1Rettungsmittel inzwischen auch ein Geschäft. Ich verkaufe Lizenzen, möchte damit Geld verdienen und investiere Zeit und Geld. Das gehört dazu. Aber trotzdem darf man nicht vergessen, woher alles kam: aus Ausbildung, aus dem Wunsch, etwas einfacher zu machen, und aus einer Idee, bei der ich selbst nicht wusste, wohin sie führen würde.

Heute kommen immer neue Menschen dazu. Neue Kunden, neue Tester, neue Gesprächspartner. Manche sagen: Ich habe da eine Idee. Andere fragen: Könnte man das nicht noch so machen? Und natürlich denke ich dann manchmal wieder: Nicht dein Ernst.

Dann gehe ich nach Hause, denke darüber nach und stelle meistens fest: Verdammt. Hat er wieder recht.

Die Reise ist also nicht vorbei. Im Mai 2027 geht es für mich das erste Mal auf die RettMobil. Allein dieser Gedanke fühlt sich noch immer ein bisschen unwirklich an: Aus einer klickbaren PDF wird plötzlich etwas, das auf einer Messe stehen soll.

Ich freue mich darauf. Und natürlich bin ich gespannt, welche Fragen dort kommen. Vermutlich genau die, an die ich vorher wieder nicht gedacht habe.

Vielleicht ist das die wichtigste Sache, die ich aus dieser ganzen Geschichte mitnehme: Ich habe 1Rettungsmittel nie alleine gebaut. Ich habe viele Stunden alleine vor dem Rechner gesessen, ja. Aber besser wurde die Idee durch Menschen.

Durch Fragen. Durch Kritik. Durch Unterstützung. Durch Kunden. Durch Freunde. Durch Kollegen. Durch meine Familie. Und ganz besonders durch eine Frau, die wahrscheinlich manchmal einfach nur in Ruhe ihren Cocktail trinken wollte und stattdessen über Jahre hinweg mitbekam, wie aus einer Idee langsam etwas Reales wurde.

Vielleicht war genau das der erste erfolgreiche Test von 1Rettungsmittel: nicht die App, sondern die Tatsache, dass jemand nah genug dran war, um die Begeisterung, die Zweifel und die vielen Umwege mit auszuhalten.

Glossar:

AAO: Alarm- und Ausrückeordnung. Sie legt fest, welche Einsatzmittel bei bestimmten Einsatzlagen alarmiert werden.

Backend: Der Teil einer Software, der im Hintergrund arbeitet, zum Beispiel Daten speichert, Lizenzen prüft oder Anfragen verarbeitet.

Frontend: Der sichtbare Teil einer Anwendung, den der Nutzer bedient.

HTML: Auszeichnungssprache, die vereinfacht gesagt die Struktur einer Webseite beschreibt.

CSS: Technik zur Gestaltung von Webseiten, zum Beispiel Farben, Abstände, Schriften und Größen.

JavaScript: Programmiersprache, mit der Webseiten und Webanwendungen interaktiv werden können.

KTW: Krankentransportwagen.

MANV: Massenanfall von Verletzten. Eine Einsatzlage mit einer größeren Zahl betroffener oder verletzter Personen, die besondere organisatorische Maßnahmen erforderlich macht.

PWA: Progressive Web App. Eine Webanwendung, die sich in vielen Punkten wie eine klassische App verhalten und teilweise offline genutzt werden kann.

QR-Code: Ein zweidimensionaler Code, der mit einer Kamera gelesen werden kann und Informationen oder Links enthalten kann.

RTH: Rettungshubschrauber.

RTW: Rettungswagen.

TestFlight: Apples Plattform zum Testen von iOS-Apps vor der öffentlichen Veröffentlichung.

Patientenablage: Ein organisierter Bereich, in dem Patienten gesammelt, weiter versorgt und für weitere Maßnahmen vorbereitet werden können.

Bereitstellungsraum: Ein festgelegter Bereich, in dem Einsatzmittel gesammelt und für ihren weiteren Einsatz geordnet bereitgehalten werden.

Ladezone: Der Bereich, in dem Patienten Transportmitteln zugeführt und für den Abtransport übernommen werden.

Sichtung: Medizinische Priorisierung von Patienten nach Behandlungs- und Transportdringlichkeit.

Stand September 2026

1Rettungsmittel und 1Transport sind aus einem konkreten Ausbildungsproblem entstanden und werden laufend weiterentwickelt.

1Control ist zum Zeitpunkt dieser Lesefassung der nächste Entwicklungsschritt. Mehrere Geräte können bereits lokal miteinander kommunizieren und auf einem zentralen Gerät live beobachtet werden. Die Funktion befindet sich weiterhin in Erprobung und Weiterentwicklung.

Die Geschichte endet deshalb bewusst nicht mit einem fertigen Produkt. Sie endet an dem Punkt, an dem sie gerade angekommen ist.

Worum es geht

Aus einer einfachen klickbaren PDF sollte eigentlich nur ein Hilfsmittel für die Ausbildung werden.

Dann kamen Fragen, Fehler, Tests, App Stores, Kunden, neue Ideen und irgendwann der Moment, in dem aus einem Nebenprojekt etwas Echtes wurde.

Dieses Buch erzählt nicht die perfekte Start-up-Geschichte. Es erzählt ehrlich, wie 1Rettungsmittel entstanden ist: mit wenig Vorwissen, viel Ausprobieren, Rückschlägen, Hilfe von anderen und der Frage, ob man einfach weitermacht, obwohl man noch nicht genau weiß, wie.

Mehr zum Projekt, zur App und zu den aktuellen Einblicken.

Homepage

Instagram

App Store

Google Play

1Rettungsmittel

Marc-André Hoffmann