

Modernisierung, die begeistert! Ein Fallbeispiel aus dem Lebensmitteleinzelhandel

Bestehende Anwendungen auf zeitgemäße, moderne Technologie-Stacks umzustellen, ist aufwendig, schafft aber eine langfristige Perspektive für die Zukunft. Ein Fallbeispiel aus dem Einzelhandel zeigt, wie sich zwei große Rich-Client-Java-Applikationen in den Browser bringen ließen. Der Artikel berichtet von den Erfahrungen mit der Onion-Architektur, von Spring und dem Web-Components-Framework von Vaadin. Er soll dazu ermutigen, die Modernisierung von Java-Anwendungen aktiv anzugehen.



Dr. Daniel Stieger
Partner



Thomas Hauser
Anwendungsentwickler



Fanny Albrecht
Softwareentwicklerin

14.07.2025

Lesezeit: 25 Minuten

👁 219 Views



Java ist seit seiner Einführung im Jahr 1996 eine der populärsten Programmiersprachen der Welt. Doch in den letzten Jahren hat sich das Tempo der Weiterentwicklung erheblich beschleunigt. Seit 2018 gibt es unter Oracle einen festen sechsmonatigen Release-Zyklus. So wird Java kontinuierlich weiterentwickelt, neue Features und Verbesserungen halten schneller Einzug. Aber auch die Community-Beteiligung mit einzelnen Entwicklern und Unternehmen führt dazu, dass neue, innovative Ideen entstehen, ausformuliert und aktiv umgesetzt werden, beispielsweise unter dem OpenJDK. Die IT-Landschaft selbst hat sich stark verändert. Cloud-Computing, Microservices und Container-Technologien erfordern neue Frameworks. Gerade auch das sehr bekannte Java-Framework Spring passt sich diesen Trends an, um Entwicklern die Arbeit zu erleichtern und die „Time to market“ zu beschleunigen. Mit jeder neuen Version werden Sicherheitslücken geschlossen und die Performance wird optimiert. In Zeiten zunehmender Cyberangriffe ist das elementar.

Refaktorisieren, austauschen, Schritt halten

Die Dynamik der schnellen Weiterentwicklung ist einer Kombination aus strukturellen Änderungen im Entwicklungsprozess, dem gestiegenen Wettbewerb, neuen technologischen Anforderungen und Aspekten der Sicherheit geschuldet. Für die bestehende, produktive Anwendungen heißt das in Konsequenz:

- zügige Aktualisierung der verwendeten Frameworks,
- Austausch von Frameworks, die auf Projekten basieren, die nicht mehr fortgeführt werden, und
- Schritt halten mit neuen technologischen Möglichkeiten, egal ob im Engineering, Deployment oder Betrieb.

Gleichzeitig geht es für Unternehmen aber vor allem darum, die bestehenden Anwendungen mit neuen „Value Added Features“ auszustatten. Man muss an jenen Features arbeiten, die für den Endanwender sichtbar sind und aus deren Perspektive den Nutzen der Anwendung erhöhen. Die kostbare Zeit der Entwickler sollte man doch für die sogenannten erlebbaren, tangiblen Features verwenden und nicht für ein aufwendiges Refactoring von Bestehendem. Austausch von Frameworks und Arbeiten an der Codebasis, das zieht am Ende des Tages doch nur Entwicklungsressourcen ab. Zum Investitionsschutz, zur Sicherstellung der langfristigen Entwicklung, müssen – oder müssten – genau für diese Tätigkeiten Entwicklungsleistung eingebracht werden. Also Hand aufs Herz – eine gute Balance zu finden, ist schwierig. Wir kennen genügend große Softwareprodukte im Enterprise-Umfeld, die erkennbar noch auf Java 6 aufbauen!

Artikel teilen



In Sammlung hinzufügen



Artikel herunterladen



Wir wollen in diesem Artikel davon berichten, wie wir ein UI Framework ausgetauscht haben, ohne die Anwendungsentwickler damit umfänglich zu beschäftigen. Dabei sind wir zusätzlich von einer Rich-Client-Architektur mit direkter Datenbankverbindung zu einer Server-zentrierten Web-Architektur gewechselt. Unsere umgebaute Enterprise-Anwendung wird jetzt bei MPREIS von etwa 500 Endanwendern simultan verwendet.

Java-Version XX und JavaFX

MPREIS ist ein österreichisches Lebensmitteleinzelhandelsunternehmen mit Sitz in Völs bei Innsbruck. Das Familienunternehmen wurde 1920 von Therese Mölk gegründet und hat sich seither zu einer bedeutenden Supermarktkette entwickelt. Das Unternehmen beschäftigt rund 6.000 Mitarbeiter. Für die Kern-Geschäftsprozesse erstellt die MPREIS IT Software größtenteils selbst. Das hat Vorteile bei der Umsetzungsgeschwindigkeit von der Idee zur verwendbaren Softwarelösung, geht aber auch mit einer sehr guten Übereinstimmung von Prozessanforderungen und Implementierung einher. Unter anderem wurde das hauseigene Filial-Warenwirtschaftssystem (eFWWS – für easy FWWS) auf der Basis von Java implementiert, ebenso die Software zum Management der digitalen Preisauszeichnungen (ePS). Die digitalen Preisauszeichnungen ersetzen mehr und mehr die konventionellen, gedruckten Preisangaben an Regalen. Beide Lösungen weisen eine beträchtliche Größe auf. Das eFWWS besteht mittlerweile aus knapp 500 Controllern mit 425 verschiedenen UIs. Neben 270 Services sind 268 DTOs, 128 Entitäten und 82 Value-Objects beteiligt.

Das ePS ist an dieser Stelle etwas kompakter. Es besteht tatsächlich aus zwei Modulen. Ein Modul bildet alle geforderten Funktionalitäten rund um das Rendern von Preisschildern ab. Hier wird auf ein altbewährtes Open-Source-Projekt, nämlich Apache-Fop, zurückgegriffen. Das zweite Modul ist für das Management der vielen Preisauszeichnungen in den Filialen zuständig. Dabei handelt es sich wieder um eine klassische Geschäftsanwendung. 150 Controller mit gut 100 UI-Seiten, 142 Services und gut 200 Datenobjekte sind notwendig. Beide Geschäftsanwendungen bauen auf dem Spring-Framework und auf JavaFX als UI-Bibliothek auf. Bisher wurden sie als Rich-Client mit direkter Datenbankverbindung pro Client unter Windows auf einer Desktop-Virtualisierungslösung ausgeliefert. Grundsätzlich läuft das sehr stabil. Die Rich Clients sind schließlich untereinander unabhängig. Allerdings benötigt die Desktop-Virtualisierungslösung für die vielen Filialen und Café-Bistros beträchtliche Ressourcen.

Begonnen wurde die Entwicklung des eFWWS 2010. Damals herrschte vonseiten des Managements große Skepsis gegenüber dem verbreiteten EJB-Standard (Enterprise JavaBeans). Das EJB-Umfeld wurde als komplex eingeordnet. Die Vielzahl von EJB-Konzepten, die damit verbundene steile Lernkurve und die strikte Architektur fanden keinen Zuspruch. Man wollte die positiven Erfahrungen, die mit relativ einfachen Rich Clients unter Linux (bereitgestellt über X-Window) gemacht wurden, in ähnlicher Weise in die Java-Welt übertragen. Daher schien ein Rich Client mit direkter Datenbankverbindung eine technologisch einfache und pragmatische Lösung zu sein. Zum Management der Abhängigkeiten kam erst Google Juice als Dependency-Injection-Framework zum Einsatz, erst später dann Spring. JavaFX galt als zukunftssichere und moderne UI-Bibliothek. Mit der Entscheidung für einen simplen Rich Client wollte man die Anwendungsentwicklung jedenfalls auf tangible Features konzentrieren. Schließlich steht die Unterstützung und Optimierung der Geschäftsprozesse im Vordergrund. Dort sind die Entwicklungsressourcen am gewinnbringendsten eingesetzt. Abbildung 1 zeigt einen Screenshot des aktuellen eFWWS in JavaFX.

FI-8990 - Wareneingang - Z-LA 30
25.03.24 - Nummern: 18835171 - Bearbeitung

Belegungen | Warenbelege | Aufgaben | Anhänge

Belegungen

Sort	Sta	Art-Nr	Bezeichnung	ist	Gründe	Gr	Partner-Bez	Prokollung	Info
0001	Offen	415147	Zewa Toilettenpapier gelb 3l 20x150	100	KAR a 3 PNC	BUR/59K	2-LA 30	---	
0002	Offen	415191	Zewa Toilettenpapier Kamille 3l 20x150		KAR a 3 PNC	BUR/59K	2-LA 30	---	
0004	Offen	416260	Hakle Topa pflegend 3l 18x140 FSC		SACK a 3 PNC	BUR/59K	2-LA 30	---	
0005	Offen	416267	Hakle Topa Kamille 3l 12x140 FSC		SACK a 3 PNC	BUR/59K	2-LA 30	---	
0006	Offen	415391	Soleo Topa Kamille 3l 8x150		SACK a 3 SACK	BUR/59K	2-LA 30	---	
0007	Offen	200173	JT Kuchertischer 3l 6x51		KAR a 4 PNC	BUR/59K	2-LA 30	---	
0008	Offen	200190	JT Toilettenpapier 10x200 4l		KAR a 7 STK	BUR/59K	2-LA 30	---	
0009	Offen	200153	JT Toilettenpapier 10x200 3l		KAR a 7 STK	BUR/59K	2-LA 30	---	
0010	Offen	200159	JT Toilettenpapier recycling 10x180 3l		KAR a 7 PNC	BUR/59K	2-LA 30	---	
0011	Offen	200227	JT Kuchertischer 3l 6x51		KAR a 4 PNC	BUR/59K	2-LA 30	---	
0012	Offen	200229	JT Toilettenpapier 10x160 4l		KAR a 7 STK	BUR/59K	2-LA 30	---	
0013	Offen	200202	JT Toilettenpapier 10x200 3l		KAR a 9 STK	BUR/59K	2-LA 30	---	
0014	Offen	200233	JT Toilettenpapier recycling 10x180 3l		KAR a 8 PNC	BUR/59K	2-LA 30	---	

Gesamt: >> 3,00 Koll (89,91 EUR) Ausgewählt: >> 3,00 Koll (89,91 EUR)

Religpositionen | Lagerpositionen

Lagerpositionen

Bereich	Typ	Stuktur	Sta	Art-Nr	Be	Typ	Grund	Pch	Gr	Whrg	Mkt-D	Geländert
Bestand	EP	-	F	415147	25.03.24 12:32	Lieferung	-	E	12,00	59K	EUR	25.03.24 12:32
Bestand	EP	-	F	415147	10.07.24 10:40	Lieferung	-	A	-3,00	59K	EUR	10.07.24 10:40

Gesamt: >> 6,00 Stück (89,91 EUR)

Abbrechen (F10) Abschließen (F12) Speichern (F12)

user: thomasbau / 2499

Abb. 1: Wareneingang eFWS-Rich-Client mit JavaFX-Technologie

15 Jahre sind nun vergangen, die Welt drehte sich weiter. Wo ein Vorteil ist, ist immer auch ein Nachteil. In puncto Sicherheit hat der Rich Client gewiss Nachteile. Das Deployment und die Verwaltung sind nicht immer einfach. Es sind aber auch die notwendigen Ressourcen der Desktop-Virtualisierung bei unserem Mengengerüst nicht mehr außer Acht zu lassen. Diese Ressourcen könnte man andernorts auch gut gebrauchen. Zudem lässt auch das, im Allgemeinen, schwindende Interesse an JavaFX Zweifel aufkommen. Die Filialen von MPREIS sind hochgradig von den beiden Lösungen eFWS und ePS abhängig. Es baut sich Druck auf. Die elektronischen Preisauszeichnungen stehen erst vor dem großen vollumfänglichen Roll-out. Müsste da nicht auch die zugehörige Managementsoftware in einer zeitgemäßen Technologie umgesetzt werden – also im Web, als Web-Anwendung?

Das Buch, das jeder kennt

Wohl jeder kennt das Buch über *Domain-Driven Design (DDD)* von Eric Evans [1], die meisten haben es vermutlich auch vollständig gelesen. Evans beschreibt darin, wie Softwareentwicklung sich an den Geschäftsdomänen orientieren sollte, um komplexe Softwaresysteme besser zu strukturieren. Der Fokus liegt auf einer engen Zusammenarbeit zwischen Entwicklern und Domänenexperten, um ein gemeinsames Verständnis zu schaffen. Mit einer einheitlichen Sprache und einem einheitlichen Modell – der Ubiquitous Language. Diese bildet eine gemeinsame Begriffswelt, soll Missverständnisse vermeiden und dann auch direkt in den Code einfließen. Daneben propagiert er Bounded Contexts, unterschiedliche Domänen-Typen und verschiedene Integrationsmuster.

Beim ersten Lesen zum Projektstart des eFWS im Jahre 2010 blieben bei uns primär die taktischen Muster hängen. Die Entitäts- und Wertobjekte; die einen mit einer eindeutigen Identität, die anderen, die bei selbem Wert austauschbar sind. Die Aggregate, die eine Gruppe von Objekten bilden und als Einheit verwaltet werden müssen. Die Repositories, die eine Abstraktion für den Zugriff auf persistente Daten bieten und damit für das Laden und Speichern von Aggregaten zuständig sind. Und die Fabriken und Services, die komplexe Aggregate erstellen und jene Geschäftslogik aufnehmen, die nicht direkt zu einer Entität gehören. Mit

diesem Instrumentarium entwickelten wir – und entwickeln noch heute – die wesentlichen Anforderungen an unsere Geschäftsanwendungen. Fabriken und Services sind in unserer Architektur jedenfalls zustandslos. Entitäten und Value-Objekte enthalten keine Infrastrukturelemente. Sie setzen sich ausschließlich aus fachlichem Code zusammen, der die Anforderungen der Domäne abbildet.

Generell haben wir uns in unserer Architektur sehr konsequent einer „Layered Architecture“ und der „Onion Architecture“ verschrieben. Beides wird im DDD-Buch von Vlad Khononov [2], das aber erst 2022 erschien, kurz und prägnant beschrieben. Den sogenannten Data-Access-Layer erfassten wir logisch durch Repositories. Den Business-Logic-Layer haben wir durch Aggregate und zustandslose Services modelliert. In diesem Layer sind alle für die Geschäftsprozesse relevanten Validierungen und Entscheidungen codiert. Schließlich haben wir in einem Top-level-Presentation-Layer die entsprechenden Benutzeroberflächen und deren Controller erfasst. Die Schichten sind streng in einem Top-Down-Kommunikationsmodell angeordnet. Konzepte des Presentation-Layers können auf den Business-Logic-Layer zugreifen, Konzepte dieses Layers wiederum auf Funktionalitäten des Data-Access-Layers. Nichts Neues. Jedenfalls haben wir umgekehrte Abhängigkeiten unterbunden, das heißt, es ist unmöglich, aus einer Entität ein Konzept der Benutzeroberfläche zu referenzieren.

Im Sinne einer Onion-Architecture mit Ports & Adapters kann man gleichzeitig davon sprechen, dass wir technische Implementierungsdetails und Infrastrukturbelange von der eigentlichen Geschäftslogik und den dafür notwendigen Datenobjekten getrennt haben. Dabei hängt die fachliche Geschäftslogik eben nicht von den technischen Implementierungsdetails ab. Die Geschäftslogik greift über Ports – in Java haben wir auf herkömmliche Interfaces zurückgegriffen – auf die notwendige Infrastruktur zurück. Tatsächlich haben wir diese Ports nur bei Top-Level-Konzepten, wie der Benutzeroberfläche und den notwendigen Controllern, benötigt. Ähnlich formuliert auch Wlaschin mit Keep I/O at the Edges [3]. Ziel ist es, die I/O-Operationen an den Rand der sogenannten Zwiebel zu bringen. So ist beispielsweise am Anfang und Ende eines Arbeitsschrittes Zugriff auf eine Datenbank notwendig, nicht während des Arbeitsschrittes. Fachlicher Code der Geschäftsanwendung braucht während des Arbeitsschrittes keinen Zugriff auf ein I/O-Port.

Diese Ideen haben wir ziemlich konsequent durchgezogen. Das ist hauptsächlich auch der parallel gestarteten Entwicklung eines Applikations-Frameworks geschuldet, das oben Beschriebenes aktiv unterstützt. Dieses Framework erlaubt die Modellierung von Benutzeroberflächen und notwendigen UI-Controllern mit einer, dem Builder-Pattern ähnlichen, Syntax direkt in Java-Klassen. Die Controller und UI-Beschreibungen sind dadurch technologieunabhängig formuliert. Das Framework nimmt diese Beschreibungen und instanziiert im Falle des Rich Clients entsprechende Pendanten in JavaFX. Schließlich stellt es einen eigenen, sehr kompakten objektrelationalen Mapper. Das Persistieren von Entitäten ist dadurch mit wenig Aufwand verbunden. Für uns hat diese strikte Aufteilung in Applikation und Applikations-Framework eine zukünftige Evolution der Software in zwei Richtungen ermöglicht. Zum einen kann die Geschäftsanwendung fachlich weiterentwickelt werden. Der Strom an neuen Anforderungen reißt ohnedies nicht ab. Zum anderen können wir die Geschäftsanwendung auch technisch evolutionieren – und zwar unabhängig vom fachlichen Code. Auch nicht ganz neu, diese Idee: *“Subject matter is the most important concern to separate out in order to create maintainable and evolvable software.”* [4]

Vor dem Hintergrund dieser Überlegungen haben wir jahrelang die Softwarelösungen pragmatisch als Rich Client ausgeliefert. Zu erkennen ist aber auch, dass wir eine gewisse Technologieunabhängigkeit und den damit verbundenen Investitionsschutz nie aus den Augen verloren haben. Sicherlich, es sammelten sich über die Jahre Altlasten in der Applikation. Auch im Applikations-Framework sind mancherorts technische Schulden entstanden. Gegen die grundsätzlichen Ideen der Architektur haben wir allerdings nicht verstoßen. Die angestrebte Modularisierung und Aufteilung „Fachliches“ und Infrastruktur haben wir nicht aufgebrochen. Ein Technologiewechsel ist damit gangbar. Und wenn man Fragen des Ressourcenbedarfs und der Skalierung ausklammert, ist auch ein Übergang vom Rich Client zu einem Server-zentrierten Betrieb direkt möglich. Das hängt in der Praxis vor allem von der Komplexität und der Größe der verwendeten Aggregate in den Applikationen ab.

Wenn JavaFX durch moderne Web-Technologie ersetzt werden soll, welchen Weg geht man dann am besten? Greift man auf die modernen JavaScript UI-Bibliotheken zurück? Sollte man dann gleich auch eine Trennung zwischen Frontend und Backend vorsehen? Müssen APIs und Microservices ins Zentrum der Überlegungen? *Und was passiert eigentlich mit dem Authentication-Flow?*

Proof of Concept

Grundsätzlich strebten wir keine Trennung entlang der Frontend/Backend-Linien an. Die Applikationslogik und Teile der Geschäftslogik formuliert in JavaScript im Browser auszuführen, das schreckte uns erst mal ab. Zudem liegen Controller, also Ablaufbeschreibungen, sowie die UI-Beschreibungen ja bereits in Java vor, ebenso wie die ganze Geschäftslogik in Java-Services und in Java-Entitäten formuliert wurden. Nach Möglichkeit wollten wir daher die gesamte Applikation serverseitig ausführen, den Applikationszustand also serverseitig halten und den Browser lediglich zur Visualisierung verwenden. Dafür gibt es im Java-Ökosystem Produkte; die [Eclipse Remote Application Platform \(RAP\)](#) zum Beispiel.

Eclipse RAP-Anwendungen werden zentral auf einem Server ausgeführt und kommunizieren über HTTP mit dem Web-Frontend. Das Programmiermodell orientiert sich vollständig an der [SWT-API \(The Standard Widget Toolkit\)](#), einer verbreiteten UI-Bibliothek. RAP stellt einen plattformübergreifenden Widget-Toolkit, der die Erstellung von Oberflächen in Java erlaubt, die Widgets selbst aber für den Browser in Web-Technologie implementiert. Die Kommunikation zwischen Web-Widget und Server-Widget, wenn man es so nennen will, übernimmt RAP. Ein erster kleiner Prototyp hat dieses Versprechen schnell illustriert. RAP bringt das Desktop-Application Feeling in den Browser. Begeistert hat uns sofort die Implementierung der Tabellenkomponente von RAP. Sowohl das eFWS als auch das ePS sind stark Tabellen-orientiert aufgebaut. Die Tabelle ist also das zentrale Mittel in der Informationsdarstellung. Die Implementierung der Tabelle in RAP ist äußerst performant. Die selektierte Zeile in einer Tabelle kann per Tastatur ohne spürbare Zeitverzögerung geändert werden. Auch Scrolling innerhalb der Tabelle unterscheidet sich in Bezug auf die schnelle Darstellung nicht von unseren Erfahrungen in der Rich-Client-Welt. Diese Aspekte hätten uns schon zu einer Go-Entscheidung geführt, wenn da nicht der Look und Feel des RAP DatePicker zu Irritationen beim Management geführt hätte. In der Tat wirkt der DatePicker nicht besonders modern. Dem RAP wurde im Zuge dessen die Zukunftsfähigkeit abgesprochen, da es im Vergleich zu anderen Web-Applikationen insgesamt zu wenig modern anfühlt.

Als nächster Framework-Kandidat rückte somit Vaadin [\[5\]](#) in das Zentrum der Aufmerksamkeit. Vaadin Ltd. ist ein finnisches Unternehmen, dessen Geschichte ins Jahr 2000 zurückreicht. Vaadin ermöglicht es Entwicklern, Web-Anwendungen vollständig in Java zu schreiben, ohne direkt mit HTML, CSS oder JavaScript arbeiten zu müssen. Bis Version 10 basierte Vaadin auf

dem Google Web Toolkit (GWT); seitdem setzt es mit dem Teilframework Flow auf Web Components. Diese neue, offene und sehr transparente Implementierung mit Web Components bringt einige Vorteile mit sich. Es lässt sich wirklich alles anpassen und individualisieren, vom Design über das Verhalten der Komponenten bis zur Integration von Web-Komponenten anderer Hersteller. Dafür ist dann allerdings schon Wissen rund um die Frontend-Technologien vorausgesetzt. Nur Java ist zu wenig. Den Möglichkeiten sind aber keine Grenzen gesetzt, im Inspektor-Fenster des Browsers lässt sich alles transparent nachvollziehen und debuggen.

Anfänglich hat uns lediglich die Umsetzung der Tabellenkomponente enttäuscht, da diese nicht als Desktop-Tabelle mit selektierter Zeile, sondern als modernes Web-Grid mit Checkboxes abgebildet ist. Etwas Recherche zeigt jedoch, dass sich dieses Grid umbauen lässt – man muss es nur tun. Auch ein Context-Menü lässt sich durch Anpassungen auf die selektierte Zeile bringen, sodass der Endanwender über diese Menüaktionen zur selektierten Zeile aufrufen kann. In puncto User-Experience hinkt diese Desktop-Tabelle der RAP-Implementierung etwas nach – unser Eindruck.

Nach einem weiteren einfachen Prototypen hat uns die Vaadin-Umgebung zugesagt. Vaadin hat sich als bedeutender Akteur im Bereich der Java-basierten Web-Entwicklung etabliert. Wir waren daher an einem wirklichen Proof of Concept interessiert. Am besten anhand der größten im Feld befindlichen Eigenentwicklung, dem eFWWS. Das Filialwarenwirtschaftssystem hat nicht nur die größte Codebasis mit den meisten UI-Seiten, es hat auch die größte Benutzerbasis in Bezug auf die Anzahl an Endanwendern, die gleichzeitig mit dem System arbeiten. Wenn es uns gelänge, diese Applikation vom Rich Client weg ins Web zu bringen, dann können wir das sicherlich mit jeder Applikation, die auf Basis unseres Applikations-Frameworks erstellt wurde. Ein wahrer Proof of Concept. Wir haben daher die Umsetzung eines „UI Adapters“ mit Vaadin 23 auf das bestehende „UI Port“ inklusive der entsprechenden Spring-Integration vorgeschlagen. Ferner haben wir uns Folgendes vorgenommen:

- Fachliche Anforderungen müssen in der Zwischenzeit ohne Unterbrechungen im eFWWS-Rich-Client implementiert werden. Die Anwendungsentwickler arbeiten weiterhin ungebrochen an geplanten Feature-Paketen für das Filialmanagement. Die Umsetzung des UI-Adapters führen wir parallel dazu aus, ohne Ressourcen der Anwendungsentwicklung zu belasten. Nach umfangreichen initialen Tests wird zusammen mit der Anwendungsentwicklung für weitere Tests eine eFWWS-Applikation mit Vaadin als Pre-Pilot deployt. Feedback der Anwendungsentwickler und eines Power-Users sind einzuarbeiten.
- Sind die ersten „Kinderkrankheiten“ in Zusammenhang mit der neuen Technologie behoben, kann eine Filiale mit der Applikation in einen Piloten starten. Im Gespräch mit den Anwendern in der Filiale ist strukturiert die Resonanz aufzunehmen. Weitere Anpassungen am UI Adapter sind zu erwarten, eventuell aber auch an der Applikation selbst. Anschließend können 5 Filialen in den Piloten übernommen werden. Für die User der Pilot-Filialen müssen die Anwendungsentwickler stets greifbar sein. Es muss proaktiv Feedback aus den Filialen erhoben werden. Nach Bewertung und Einarbeitung von Unzulänglichkeiten und Fehlern kann die Zahl der Filialen im Piloten verdoppelt werden. Diese Roll-out-Strategie ist grundsätzlich fortzuführen, bis alle Filialen umgestellt sind.
- Eine gute Abstimmung zwischen den Teams (Infrastruktur, Windows, App-Framework und Applikation) ist durch regelmäßige Meetings sicherzustellen. Deployt wird immer der herkömmliche Rich Client und der neue Vaadin-Client, beides auf aktueller Anwendungsversion. Damit können auch Pilot-Filialen im Notfall auf den herkömmlichen Client wechseln. Der Browser für den Vaadin-Client wird bis auf Weiteres in der Desktop-Virtualisierung der Filialen ausgeführt.

- Die Endanwender sollen den Technologiewechsel nach Möglichkeit nicht wahrnehmen. Im Tagesgeschäft und dem kontinuierlichen Betrieb liegt der Fokus. Ohne zusätzliche Erklärungen oder Unterweisungen sollen die gewohnten Funktionalitäten zur Verfügung stehen. Das heißt, dass sich beispielsweise tiefere Menüstrukturen der Applikation nicht verändern dürfen, ebenso wenig wie der Aufbau der UI-Seiten selbst. Die Endanwender sind mit dem eFWWS vertraut und haben schließlich Erwartungen, an welcher Stelle welche Informationen zu finden sind. Anpassungen an der Startseite oder dem Top-Level-Menü sind aber sicher zumutbar. Ein ansprechendes, modernes Design und leichte Änderungen in der Informationsdarstellung sind möglich und gewünscht, die grundlegende Struktur der bekannten Oberfläche ist beizubehalten.
- Die Anwendungsentwickler selbst haben zusätzliche Goodies, also kleinere Features, die Begeisterung beim Endanwender auslösen können und Neuartigkeitscharakter haben, einrekamiert: Fähigkeiten für ein Multi-Version-Deployment auf Tomcat, Verbesserungen in der Darstellung von Medien, allen voran einer Bildvorschau; neue Möglichkeiten für das Erstellen von Grafiken oder ein Single-Sign-on statt herkömmlicher Anmeldung – das alles ist im Web-Umfeld natürlich überhaupt kein Problem. Im Gegenzug würden sie die gewachsenen Bereiche der Applikationen nochmals untersuchen und vereinheitlichen, sodass bei beiden Applikationen überall dieselben Bedienungsmuster verwendet werden.

Vom Management gab es für dieses Vorhaben dann ein finales Go, mit der unbedingten Zielsetzung, die Desktop-Virtualisierung für Filialen gesamthaft abzulösen. Dazu müssen sowohl das eFWWS als auch das ePS mittelfristig in den Web-Browser migriert werden. Eine zentrale Ausführung von Geschäftslogik auf einem Server mit zentralisiertem Datenbankzugriff ist anzustreben. Der Proof of Concept erfolgt direkt anhand der Größeren der beiden Applikationen. Dies soll Lerneffekte auf dem Weg der Migration maximieren und den Umstieg mit der zweiten Applikation überhaupt risikofrei stellen.

Gesagt getan – Implementierung nach Plan

Grundsätzlich existierte das UI-Port als Element in unserem Applikations-Framework bereits. Es wurden am Port natürlich einige Anpassungen notwendig, die durch den UI-Adapter für Vaadin entstanden. Insgesamt besteht das Port nun aus 16 verschiedenen Java-Interfaces. Die Implementierung des UI-Adapters mit zusätzlichen Features, die nicht das Port betreffen (beispielsweise SSO), resultierte in knapp 90 Java-Klassen mit 6000 Zeilen Code. Ergänzend waren etwa 500 Zeilen JavaScript notwendig, das Design wurde in 14 CSS-Dateien mit insgesamt 550 Zeilen implementiert.

Schwerwiegende Veränderungen am Applikations-Framework waren nicht notwendig. Repositories und Services sind zustandslos. Mit dem Spring IOC Container werden sie instanziiert, sowohl im Falle des Rich Clients als auch im Falle des Web-Clients. Applikationsinstanzen, Controller und UI-Seiten werden dynamisch zur Laufzeit erzeugt. Vermutlich hat der frühe Einsatz des Spring-Frameworks im Rich Client gewissermaßen eine Multi-User kompatible Architektur für einen Serverbetrieb geprägt.

Eine externe Firma entwickelte das Design für den UI-Adapter mit der Vorgabe, sich stark am bestehenden Rich Client zu orientieren. Es entstand ein – man könnte sagen „nüchternes“ – Design mit wenig Farben, insgesamt eher hell, was die Lesbarkeit erhöhen soll. Es sind aber auch Elemente des modernen Web-Designs in den Details eingeflossen. So werden Stati, ein häufiges Element unserer Applikationen, durch Chips visualisiert. Daneben führte der externe Input auch zu Optimierungen in der Informationsdarstellung. Auf Hauptseiten verwenden wir keine deaktivierten Formulare zur Informationsdarstellung, sondern einen linksbündigen Text in Form einer Hauptüberschrift und einer kleiner gehaltenen Subüberschrift. Beide Überschriften

können mehrere Informationselemente enthalten, die durch einen großen Punkt getrennt werden. Im Grunde genommen eine kleine Änderung. Sie lässt die Applikation jedoch moderner erscheinen und spart erkennbar Platz am Bildschirm. Ehrlicherweise müssen wir festhalten, dass diese Anpassung von den Applikationsentwicklern in der Applikation selbst durchgeführt wurde – im Zuge des Applikations-Reviews. Dabei haben die Applikationsentwickler zusätzliche Bedienungsmustern diskutiert, standardisiert und eingeführt. Der Aufwand hielt sich in Grenzen, insofern, als dass er nicht im Besonderen wahrgenommen wurde.

Die angestrebte Einführung des eFWWS verlief äußerst zufriedenstellend. Etwas länger gestalteten sich die ersten Feedback-Zyklen, die zum Teil umfangreiche Anpassungen am UI-Adapter nach sich zogen. Unter anderem fiel einem Zyklus auch die besondere Schriftart der neuen Oberfläche zum Opfer. Zugunsten besserer Lesbarkeit ist jetzt Standard-Helvetica im Einsatz. Schneller gestaltete sich hingegen der Roll-out in den Filialen. Die Implementierung lief von Anfang an sehr stabil. Nach den initialen Feedback-Runden und dem Einsatz in 50 Filialen wurde dann gleich auf 150 ausgeweitet, nach erneuter kurzer Prüfung dann auf alle Filialen. Dabei stellte das Management zu keinem Zeitpunkt den Roll-out als großes, innovatives Projekt dar. Ganz im Gegenteil: Kommuniziert wurde die neue eFWWS Oberfläche nur als technische Notwendigkeit, die weder Arbeitsabläufe noch Änderungen an der Bedienung nach sich ziehen. Abbildung 2 zeigt dasselbe UI des eFWWS, das wir eingangs gezeigt haben in Web-Technologie. Ausgeführt wird das eFWWS im App-Mode des Browsers.

FI-8990 • Wareneingang • Z-LA 30
25.03.24 • Nummer: 18835171 • Bearbeitung

Bewegungen Warenbelege Aufgaben Anhänge

Sort	Sta	Art.Nr.	Ist. Gebinde	7 Eh	Partner-Bez	Prüfung	Info
0001	Offen	415147 Zewa Toilettenpapier weiß 3l 20x150	3 Stk KAR a 3 PKG	9 EUR/Stk	Z-LA 30	---	
0002	Offen	415145 Zewa Toilettenpapier gelb 3l 20x150	KAR a 3 PKG	9 EUR/Stk	Z-LA 30	---	
0003	Offen	415191 Zewa Toilettenpapier Kamille 3l 20x150	KAR a 3 PKG	9 EUR/Stk	Z-LA 30	---	
0004	Offen	416268 Hako Topo pflegend 3l 10x140 F SC	SACK a 3 PKG	4 EUR/Stk	Z-LA 30	---	
0005	Offen	416267 Hako Topo Kamille 3l 10x140 F SC	SACK a 3 PKG	5 EUR/Stk	Z-LA 30	---	
0006	Offen	415391 Sckel Topo Kamille 3l 8x150	SACK a 8 SACK	1 EUR/Stk	Z-LA 30	---	
0007	Offen	200173 JT Küchenpapier 3l 6x51	KAR a 4 PKG	3 EUR/Stk	Z-LA 30	---	
0008	Offen	200190 JT Toilettenpapier 10x160 4l	KAR a 7 STK	4 EUR/Stk	Z-LA 30	---	
0009	Offen	200155 JT Toilettenpapier 10x200 3l	KAR a 7 STK	3 EUR/Stk	Z-LA 30	---	
0010	Offen	200155 JT Toilettenpapier recycling 10x18	KAR a 7 PKG	2 EUR/Stk	Z-LA 30	---	
0011	Offen	200237 JT Küchenpapier 3l 6x51	KAR a 4 PKG	2 EUR/Stk	Z-LA 30	---	
0012	Offen	200229 JT Toilettenpapier 10x160 4l	KAR a 7 STK	2 EUR/Stk	Z-LA 30	---	
0013	Offen	200202 JT Toilettenpapier 10x200 3l	KAR a 9 STK	2 EUR/Stk	Z-LA 30	---	
0014	Offen	200233 JT Toilettenpapier recycling 10x18	KAR a 8 PKG	2 EUR/Stk	Z-LA 30	---	

Gesamt: >> 3,00 Koll (89,91 EUR) Ausgewählt: >> 3,00 Koll (89,91 EUR)

Belegpositionen Lagerpositionen

Bereich	Typ	Stu...	Sta	Art.Nr.	1. Bis	Typ	Grund	-	Peh Eh	1. Wtng	MHD	Geändert
Bestand	EP	-	F	415147	2. 25.03.24 12.32	Lieferung	-	E	12.00	Stk	1. EUR	25.03.24 12.32
Bestand	EP	-	F	415147	1. 10.07.24 10.40	Lieferung	-	A	-3.00	Stk	-. EUR	10.07.24 10.40

Gesamt: >> 9,00 Stück (89,91 EUR)

< Abbrechen (ESC) Abschließen (F10) Speichern (F12)

Abb. 2: eFWWS – Wareneingang in Web-Technologie

Überrascht sind wir vom serverseitigen Ressourcenbedarf, denn gerade bei der Frage der Skalierbarkeit hielten sich trotz anfänglicher Prüfung die Zweifel hartnäckig. Einerseits ist es der Ressourcenbedarf von Vaadin, dessen Java-Implementierung der Komponenten-API und der eigentlichen Real-time-Kommunikation mit den Browser Web Components, zum anderen sind es die Erfordernisse unseres Applikations-Frameworks und dann vor allem der Applikation. Komplexe und große Aggregate, die von der Datenbank geladen werden, benötigen bei vielen simultanen Anwendern entsprechend Speicher. Bei allem halten wir den Applikationszustand eines Anwenders vollständig auf dem Server, nicht im Browser, wie mit den modernen JS-

Frameworks und der Frontend/Backend-Trennung üblich. Betrieben wird das Ganze auf einem Apache Tomcat. Dieser Open-Source-Anwendungsserver selbst war uns nicht neu, er wird im Unternehmen bereits in der Breite eingesetzt.

Wir können also von nichts Aufregendem berichten! Zu Stoßzeiten bedient ein Tomcat gleichzeitig an die 500 Endanwender mit Vaadin, das heißt, 500 Anwendungsinstanzen werden von Tomcat gehalten. Unter Linux haben wir der JVM mit Tomcat bis zu 7 GByte Speicher zugesichert. Tatsächlich beläuft sich die Heap-Größe während des Betriebes auf 2,5 GByte, nach einer erzwungenen Garbage Collection 1 GByte! Durchschnittlich sind vielleicht an die 200 Threads zu beobachten, die CPU-Auslastung ist unter einem Prozent, in einzelnen Spitzen 30 Prozent. Im Vergleich dazu mussten wir Jahre zuvor bei selber Applikation den maximal verfügbaren Speicher des Rich Clients unter Windows auf 1,5 GByte anheben. Es gab einzelne Spitzen, die einen OutOfMemoryError nach sich zogen. Das führte zu Diskussionen mit Blick auf den verfügbaren Speicher der Desktop-Virtualisierungsmaschinen. Beim eFWWS im Browser weist der Windows Taskmanager nun 200 MByte Speicherverbrauch des Browsers aus. Insgesamt sind wir mit diesen Zahlen weit unter dem, was wir uns erwartet hätten.

Lessons learned

Nach einem halben Jahr Betrieb haben wir bisher keine Schwachstellen finden können. Den Proof of Concept mit der größten Applikation, der Filialwarenwirtschaft, konnten wir positiv abschließen. Zwei Monate später, mit Ende 2024, haben wir auch das ePS umgestellt. Von Beginn an lief die Software in Web-Technologie äußerst stabil. Bisher haben wir kein Load-Balancing mit mehreren Tomcats installiert. Für beide Applikationen ist ein Tomcat mit einer JVM in Betrieb – mit drastischen Konsequenzen im Falle eines JVM-Crash. Ein zweiter Tomcat bedient an die 10 Filialen mit in Pilot befindlicher Applikationsversion. Wir können im Sinne eines Continuous Delivery gleichzeitig mehrere Versionen der Applikation bereitstellen und auch während der Filialarbeitszeiten einfach ein Versions-Upgrade durchführen. Wir fühlen uns gut gerüstet für den Umstieg auf neue Java-Versionen und Adaptierungen, die im Zuge eines Framework-Austausches notwendig werden.

Die Arbeitsteilung, die unser Applikations-Framework erlaubt, hat sich bewährt. Mit konzentriertem Blick auf die Geschäftsprozesse und den Endanwender konnten Applikationsentwickler uneingeschränkt neue fachliche Features und Funktionen implementieren. Parallel dazu haben wir mit einem externen Entwickler das Applikations-Framework durch einen Vaadin UI-Adapter ergänzt und erforderliche Anpassungen vorgenommen. Einen Großteil der Tests konnten wir ohne Beteiligung der Applikationsentwickler durchführen, für den wirklichen Proof of Concept wurden sie natürlich eingebunden. Prinzipiell wären keine Änderungen am Applikationscode (beziehungsweise an den Modellbeschreibungen) erforderlich gewesen. Dennoch kam es im Zuge der Umstellung auf einen Web-Client zu Refactorings von Bedienungsmustern. Davon haben die Applikationen profitiert.

Für das Management ist in Summe der Umstieg ein Vorzeigeprojekt. Das Applikations-Framework wurde gestärkt und ausgebaut. In Zukunft können alle auf dem Framework basierenden Softwarelösungen mit zukunftsorientierter Web-Technologie ausgeliefert werden. Applikationsentwickler können dabei konstant und ohne Unterbrechung fachliche Anforderungen umsetzen. Diese Entwicklungsressourcen sind weitestgehend von Technologieänderungen geschützt. Die Desktop-Virtualisierung für Filialen wird abgebaut und anderweitig eingesetzt. Einige Goodies sind auch für Endanwender sichtbar.

Aus Perspektive der Endanwender hat sich nicht allzu viel verändert – was die Abbildung 3 und 4 zeigen.

Filiale-Nummer	Filiale-Bez	Alt-ID	Typ	Subtyp	Bel-Dat	Referenz	Status	Ver	Info	Hist
8990	## MPR Referenz...	22436302	Best-Kont	Best-Check	04.03.25	BC-20250304	Geprüft	0	BC-20250304	Prod.
8990	## MPR Referenz...	22430337	Best-Kont	Best-Check	03.03.25	BC-20250303	Geprüft	0	BC-20250303	Prod.
8990	## MPR Referenz...	22343006	Best-Kont	Kont-Zahl	24.02.25	KZ-20250224	Geprüft	0	Zahlenerhebung	Prod.
8990	## MPR Referenz...	22217762	Best-Kont	Best-Check	12.02.25	BC-20250212	Geprüft	0	BC-20250212	Prod.
8990	## MPR Referenz...	22159941	Best-Kont	Kont-Zahl	06.02.25	KZ-20250206	Geprüft	0	Zahlenerhebung	Prod.

Abb. 3: Bestandskontrolle eFWS-Rich-Client in Java FX-Technologie

Filiale-Nummer	Filiale-Bez	Alt-ID	Typ	Subtyp	Bel-Dat	Referenz	Status	Ver	Info	Hist
8990	## MPR Referenz...	22436302	Best-Kont	Best-Check	04.03.25	BC-20250304	Geprüft	0	BC-20250304	Prod.
8990	## MPR Referenz...	22430337	Best-Kont	Best-Check	03.03.25	BC-20250303	Geprüft	0	BC-20250303	Prod.
8990	## MPR Referenz...	22343006	Best-Kont	Kont-Zahl	24.02.25	KZ-20250224	Geprüft	0	Zahlenerhebung	Prod.
8990	## MPR Referenz...	22217762	Best-Kont	Best-Check	12.02.25	BC-20250212	Geprüft	0	BC-20250212	Prod.
8990	## MPR Referenz...	22159941	Best-Kont	Kont-Zahl	06.02.25	KZ-20250206	Geprüft	0	Zahlenerhebung	Prod.

Abb. 4: Bestandskontrolle eFWS in Web-Technologie

Die Funktionalitäten des Rich Clients sind natürlich 1:1 auch im Web-Client nutzbar. Es sind aber keine großen Änderungen in der Benutzeroberfläche mit der Menüstruktur erkennbar. Damit finden sie alles am gewohnten Ort. Die Accessibility der Software dürfte allein durch

Verwendung eines herkömmlichen Browsers gestiegen sein. So ist beispielsweise durch den Browser auch ein Zoom möglich, um die Lesbarkeit nochmals zu erhöhen. Ein Login ist durch Single-Sign-on nicht mehr notwendig, ebenso wenig wie Schulungen für die Endanwender. Auch die Dokumentation der Applikationen ist noch völlig korrekt, wenn man vom Design der Screenshots absieht. Diese sind wohl nach und nach zu tauschen. Man könnte sagen, dass die Endanwender „weitestgehend ruhig sind“ – beim Roll-out von Neuem vielleicht gar kein schlechtes Zeichen.

Die „Layered Architecture“ und die „Onion Architecture“ haben sich extrem bewährt und diesen Umstieg erst ermöglicht. Wir hätten tatsächlich nach 15 Jahren Applikationsentwicklung im eFWS keine Zeile der Anwendung ändern müssen. Es ist uns gelungen, nicht gegen die festgelegte Architektur zu verstoßen. Wie wir in einem früheren Artikel [6] bereits berichtet haben, setzen wir teilweise auf Model-Driven Software-Development (MDS) und verwenden Codegeneratoren in **JetBrains MPS** [Meta Programming System]. In der Praxis schränken wir so den Zugriff der Applikationsentwickler auf die unterliegende Infrastruktur und die verwendeten Frameworks ein. Sicherlich hat das auch Nachteile in der Flexibilität, es erhöht insgesamt aber die Standardisierung und die Entwicklungsgeschwindigkeit und eben auch die Möglichkeiten bei einer anstehenden Softwaremodernisierung.

Literaturangaben

[1] E. Evans, Domain-Driven Design, Addison-Wesley, 2004

[2] V. Khononov, Learning Domain-Driven Design, O'Reilly, 2022

[3] S. Wlaschin, Domain Modeling Made Functional, Pragmatic Bookshelf, 2018

[4] M. Völter, How to understand almost anything, Version 2, Eigenverlag, S. 15, 2024

[5] Book of Vadin, <https://vaadin.com/book> (Abgerufen am 25. März 2025)

[6] D. Stieger, W. Messner, M. Molk, B. Flunger, Filial-Warenwirtschaftssystem auf Basis von Java und drei DSLs in JetBrains MPS, in: JavaSPEKTRUM, 2/2023, siehe: <https://www.sigs.de/artikel/filial-warenwirtschaftssystem-auf-basis-von-java-und-drei-dsls-in-jetbrains-mps/>