



Probe-IPA

Eingehende Message-Queue-Nachrichten im Web-UI

Probe-IPA von Joel Vontobel

Ergon Informatik AG
20. November 2024

Inhalt

I	Umfeld und Ablauf	4
1	Aufgabenstellung	5
1.1	Ausgangslage	5
1.2	Detaillierte Aufgabenstellung	5
1.3	Mittel und Methoden	6
1.4	Vorkenntnisse	6
1.5	Vorarbeiten	6
1.6	Neue Lerninhalte	7
1.7	Arbeiten in den letzten 6 Monaten	7
2	Projektaufbauorganisation	8
3	Benützte Firmenstandards	9
4	Arbeitsumgebung	10
4.1	Arbeitsplatz	10
4.1.1	Office Arbeitsplatz	10
4.2	Verwendete Tools	11
5	Versionierung und Sicherung der Arbeitsergebnisse	12
5.1	Verwendung von Git zu Versionierung	12
5.2	Quellcode	12
5.3	Probe-IPA Dokumentation	13
6	Projektmanagementmethode	15
6.1	IPERKA	15
6.2	Alternative Methode	16
7	Arbeitsprotokoll	17
II	Projekt	28
8	Kurzfassung	29
9	Informieren	30
9.1	Projektumfeld	30
9.1.1	Einsatzzweck	30
9.1.2	Funktion	30

9.2	Anforderungen	32
9.2.1	Minimalanforderungen	32
9.2.2	Erweiterung: Pagination	33
9.2.3	Erweiterung: Filter	34
10	Planen	36
10.1	Arbeitspakete	36
10.1.1	Informieren	37
10.1.2	Planen	37
10.1.3	Entscheiden	38
10.1.4	Realisieren	39
10.1.5	Kontrollieren	40
10.1.6	Auswerten	40
10.1.7	Rahmenaufgaben	41
10.2	Lösungskonzept für die Struktur vom Backend	43
10.2.1	Erstellung der Endpunkte	43
10.2.1.1	DAO-Klassen	43
10.2.1.2	Klassendiagramm für das Backend	43
10.2.2	Verwendete Technologien	44
10.3	Lösungskonzept für die Struktur vom Frontend	44
10.3.1	Bestehende Implementationen	44
10.3.2	Neue Implementationen	45
10.3.3	Verwendete Technologien	45
10.4	Entscheidung der Erweiterung	45
10.4.1	Pagination	45
10.4.2	Filter	46
10.4.3	Entscheid	46
10.5	Lösungskonzept für die Struktur vom Filter	46
10.5.1	Bestehende Implementation	46
10.5.2	Neue Implementationen	46
10.5.2.1	MQ_IN_STATUS	46
10.5.3	Begriffe im Nachrichten Inhalt Filtern	47
10.5.4	Sequenzdiagramm für die Erweiterung Filter	47
10.6	Testkonzept	48
10.6.1	Benötigte Testmittel	48
10.6.2	Automatisierte Tests	49
10.7	Manuelle Tests	49
10.8	Testfälle	50
10.8.1	Testfälle der Mindestanforderungen	50
10.8.2	Testfälle der Erweiterung Filter	52
11	Entscheiden	54
11.1	Filterung der Daten auf dem Server	54
11.1.1	Vorteile	54
11.1.2	Nachteile	54
11.2	Filterung der Daten auf der Datenbank	55
11.2.1	Vorteile	55

11.2.2	Nachteile	55
11.3	Entscheidung	55
12	Realisieren	56
12.1	Endpoints für die Mindestanforderungen erstellen	56
12.1.1	MqInDTO	56
12.1.2	MqInMapper	57
12.1.3	MqInService	57
12.1.4	MqInServiceImpl	57
12.1.5	MqTableDao	57
12.1.6	MqTableHibernateDao	58
12.1.7	MqInResource	58
12.2	Die Seite und Tabelle im Frontend erstellen	59
12.2.1	Erstellung der Tabelle	59
12.2.2	Anzeigen der ganzen Nachricht	60
12.2.3	Erneutes Ausführen vom Verarbeitungsversuch	60
12.3	Endpoints für die Erweiterung Filter erstellen	61
12.3.1	MqInFilterDto	61
12.3.2	MqInService	61
12.3.3	MqInServiceImpl	62
12.3.4	MqTableDao	62
12.3.5	MqTableHibernateDao	62
12.4	Die Erweiterung in die Tabelle integrieren	63
12.4.1	mq-in-filter-url	63
12.4.2	mq-in-filter.component	64
12.4.3	Fehlerbeschreibung	65
12.5	ReleaseNotes schreiben	65
13	Kontrollieren	66
13.1	Allgemein	66
13.2	Integration-Tests	66
13.2.1	testGetAllMqIns	66
13.2.2	testExecuteNewProcessingAttempt	67
13.2.3	testGetFilteredMqInEntries	67
13.3	Unit-Tests	68
13.3.1	testMappingWithSinglePC	68
13.3.2	testMappingWithMultiplePCs	68
13.3.3	testMappingOfMqInFilterDto	69
13.4	Codequalität prüfen	69
13.4.1	Falscher Rückgabewert bei setMqInStatusToNew	70
13.4.2	JUnit-Test Klassen müssen final sein	70
13.4.3	LocalDateTime / -Millis.class	70
13.4.4	MqInMocks	70
14	Auswerten	71
	Glossar	72

Abbildungsverzeichnis	72
Quellenverzeichnis	73

Teil I

Umfeld und Ablauf

1 Aufgabenstellung

In diesem Kapitel ist die Aufgabenstellung der Probe-IPA aufgeführt. Die Inhalte wurden zu einem grossen Teil von der originalen Aufgabenstellung übernommen und Angepasst.

1.1 Ausgangslage

Die Firma Ergon Informatik AG entwickelt sein einigen Jahren eine Transaktions-Autorisierungs-Lösung für einige Banken in der Schweiz. Dieses Projekt heisst CardX und wird durch ein 12 zwölfköpfiges Team umgesetzt. In diesem Projekt ist der Lernende seit Januar 2024 tätig und kennt sich deshalb schon ein bisschen aus.

Das Projekt kommuniziert mit verschiedenen bankspezifischen IT-Systemen wie zum Beispiel dem Kernbankensystem oder Service-Büros über Message-Queues. Diese Message-Queues werden in der Datenbank-Tabelle `MQ_TABLE` (eingehende Nachrichten) und `MQ_OUT` (ausgehende Nachrichten) zwischengespeichert. Falls man diese Message-Queues anschauen oder bearbeiten möchte, muss man dies in der Datenbank machen. Weil das ziemlich umständlich ist, besteht die Aufgabe des Lernenden jetzt daraus diese Message-Queues in einem Web-GUI darzustellen und sinnvolle Interaktionen mit diesen Daten anzubieten.

1.2 Detaillierte Aufgabenstellung

Minimalanforderungen Das Ziel dieser Aufgabe ist es, den Inhalt der Tabelle `MQ_TABLE` in diesem Web-GUI sichtbar zu machen und dem Nutzer sinnvolle Interaktionen mit diesen Daten anzubieten.

Es soll im Web-GUI eine neue Seite erstellt werden mit dem Inhalt einer Tabelle, welche die Message-Queues abbilden soll. Die Tabelle soll eine Hand voll Spalten besitzen, sodass sie übersichtlicher ist. Die Seite soll stimmig in das UI eingebaut werden. Im Backend sollen die neuen Methoden mithilfe von Unit-Tests abgedeckt werden.

Zusätzlich kann der Lernende noch zwischen zwei Erweiterungen entscheiden, welche er implementieren möchte.

Erweiterung: Pagination Bei der ersten Erweiterungen ist das Ziel ein Paginator zur Tabelle hinzuzufügen. Eine Pagination ist wenn man eine Liste oder Tabelle auf ein paar Einträge limitiert und anschliessend weitere Einträge anzeigen kann mit einer Pfeiltaste. Dies hilft, das Laden der Seite zu verkürzen, da die Einträge, die nicht angezeigt werden, erst geladen werden, wenn sie auch wirklich gebraucht werden.

Erweiterung: Filter Die zweite Erweiterung ist ein Filtersystem. Die Seite soll nach dem Status, Inhalt der Nachricht und dem Datum gefiltert werden können. Die Filter-Werte sollen ausserdem in der URL Abgebildet werden, um das Teilen von gefilterten Ergebnissen zu vereinfachen oder den Filter als Lesezeichen ablegen zu können.

1.3 Mittel und Methoden

Technologien

- SQL
- Java
- TypeScript
- HTML
- Angular

Tools

- IntelliJ (IDE)
- Docker
- Bitbucket
- Confluence
- Jira
- Postman

1.4 Vorkenntnisse

Der Lernende hat bereits viele Arbeiten im Projekt CardX gemacht. Unter anderem im Backend und an CardX-spezifischen Tools. Die Codebasis hat der Lernende in den letzten 9 Monaten gut kennengelernt und findet sich gut zurecht. Der Lernende hat auch bereits die Tabelle TASK von der Datenbank in das Web-GUI gebracht, was eine ähnliche Aufgabe war wie die jetzige Aufgabenstellung.

Durch die früheren Projekte, wie ein Fussballtippspiel und eine Anmelde-Plattform für Bewerbende, konnte er bereits viel Erfahrung mit Java und Angular sammeln.

1.5 Vorarbeiten

Durch das bereits existierende Projekt und die vielen Arbeiten, die der Lernende bereits gemacht hat, musste er keine Vorarbeiten leisten.

1.6 Neue Lerninhalte

- Pagination:

Mit Pagination hat der Lernende sich noch nie auseinandergesetzt. Er hat es schon oft auf anderen Seiten gesehen aber noch nie selbst implementiert.

- Filtersystem:

Im Projekt WM-Tippspiel gab es ein Filtersystem, aber dieses hat der Lernende nicht selbst implementiert und ist so ein neuer Lerninhalt.

1.7 Arbeiten in den letzten 6 Monaten

In den letzten 6 Monaten hat der Lernende sich, wie oben schon genannt, mit dem Projekt CardX auseinandergesetzt und ist ein aktives Team Mitglied. Er hat viele verschiedene Arbeiten umgesetzt. Einige davon hier:

- CheckDB Task:

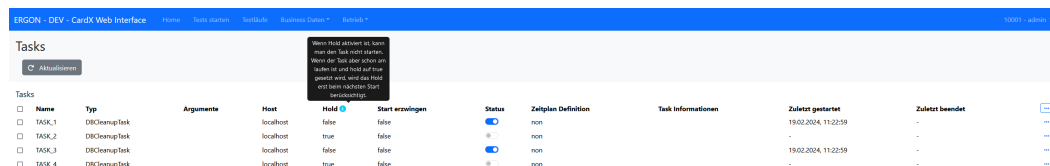
In dieser Aufgabe geht es darum, eine Aufgabe zu erstellen, welche periodisch oder manuell ausgeführt werden kann. Diese Aufgabe überprüft die Datenbankdefinition, ob immer noch alles fehlerfrei ist. Es werden Sequenzen, Indexe, Felder und mehr in eine Datei geschrieben, gespeichert und anschliessend mit der vorherigen Datei auf Veränderungen verglichen. Bei einer Veränderung schlägt die Aufgabe fehl und die Differenz wird in der Fehlermeldung angezeigt.

- ServiceLogMove und Zip:

Mit der Aufgabe ServiceLogMerge werden Protokolle des Systems in Dateien geschrieben und gespeichert. Ein Teil dieser Dateien wird jetzt mithilfe der Aufgabe in bestimmte Verzeichnisse verschoben und komprimiert. Die Dateien werden nach Bank sortiert und anschliessend in ihn vorgesehenes Verzeichnis verschoben. Auch diese Aufgabe wird periodisch jeden Tag ausgeführt, um die Dateiablage übersichtlich zu halten.

- Tasks im Frontend anzeigen und bearbeiten:

Diese Aufgabe zeigt alle Tasks im Frontend an und man kann sie dort auch bearbeiten. Das Ziel von dieser Aufgabe war gleich wie die Mindestanforderungen von der Aufgabenstellung.



Name	Type	Argumente	Host	Held	Start erzeugen	Status	Zeitplan Definition	Task Informationen	Zuletzt gestartet	Zuletzt beendet
TASK_1	DBCleanupTask		localhost	false	false	on	non		19.02.2024, 11:22:59	-
TASK_2	DBCleanupTask		localhost	true	false	on	non		-	-
TASK_3	DBCleanupTask		localhost	false	false	on	non		19.02.2024, 11:22:59	-
TASK_4	DBCleanupTask		localhost	true	false	on	non		-	-

Abbildung 1.1: Anzeigen und bearbeiten der Tasks

2 Projektaufbauorganisation

In der folgenden Tabelle sind die an der Probe-IPA beteiligten Personen und ihre jeweiligen Aufgaben aufgeführt.

Person	Rolle	Aufgabe/Verantwortung
Joel Vontobel	Kandidat (K)	Umsetzen der Facharbeit
Loris Diana und Dominic Monzón	Verantwortliche Fachkraft (VF)	Facharbeit begleiten, technische Fragen beantworten, Bewertung der Facharbeit
Bernd Lienberger	Hauptexperte (HEX)	IPA bezogene Fragen beantworten, Entscheiden bei auftretenden Problemen, Besuchstermine festlegen, Fachgespräch leiten, Bewertung der Facharbeit
	Nebenexperte (NEX)	Notizen erstellen zu Präsentation und zum Fachgespräch, Bewertung der Facharbeit

3 Benützte Firmenstandards

Es werden keine spezifischen Firmenstandards verwendet.

4 Arbeitsumgebung

In diesem Kapitel ist beschrieben, wie die Arbeitsumgebung des Lernenden während der Probe-IPA aussah.

4.1 Arbeitsplatz

4.1.1 Office Arbeitsplatz

Die Probe-IPA wird am gewohnten Arbeitsplatz im Fünferbüro des Lernenden durchgeführt. Als Arbeitsgerät wird ein Notebook verwendet, welches mithilfe einer Dockingstation das Gerät mit zwei Monitoren und dem Firmennetzwerk verbindet. Der Stuhl und Tisch sind höhenverstellbar, und der Lernende kann dadurch in verschiedenen Sitzpositionen oder stehend arbeiten.

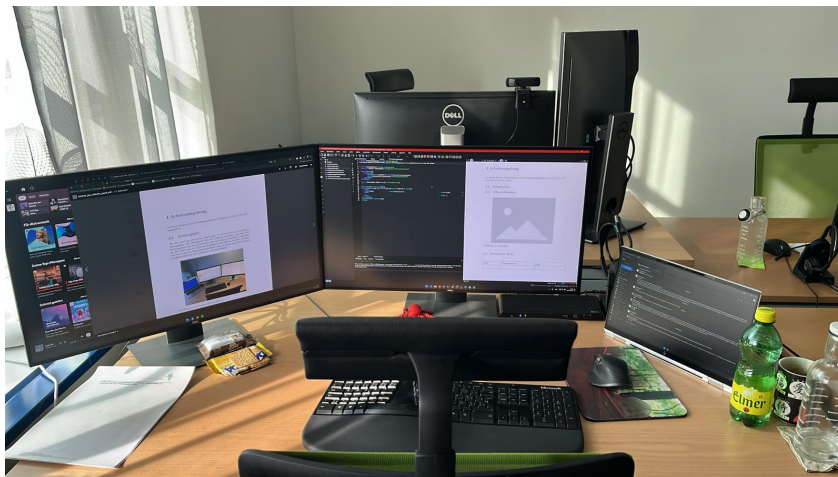


Abbildung 4.1: Arbeitsplatz des Lernenden

4.2 Verwendete Tools

Die folgende Tabelle zeigt auf, welche Tools für die Umsetzung der Probe-IPA eingesetzt wurden.

Tool	Einsatzzweck	Link
IntelliJ	Entwicklungsumgebung für die Programmierung	https://www.jetbrains.com/de-de/idea/
Docker	Ausführen der Programme	https://www.docker.com/
Git	Versionierung vom Quellcode	https://git-scm.com/
Postman	Ausführen von HTTP-Requests (Testen vom Bankend)	https://www.postman.com/
Bitbucket	Speicherung der Quellcodes	https://bitbucket.org/product/
Jenkins	Tool für Pipelines um Tests, Codequalität automatisch zu prüfen	https://www.jenkins.io/
Confluence	Probe-IPA Kriterien	https://www.atlassian.com/de/software/confluence
Jira	Aufgabenstellung	https://www.atlassian.com/de/software/jira
Draw.io	Erstellen von Diagrammen und Abbildungen	https://www.drawio.com/
TexStudio	Dokumentationstool	https://www.texstudio.org/
LaTeX	Ein Dokumentenvorbereitungssystem	https://www.latex-project.org/
Mattermost	Text basiertes Kommunikationsmittel	https://mattermost.com/
Microsoft Teams	Video basiertes Kommunikationsmittel	https://www.microsoft.com/de-ch/microsoft-teams/group-chat-software
Microsoft Excel	Erstellung und Bearbeitung des Zeitplans	https://www.microsoft.com/de-ch/microsoft-365/excel?market=ch

5 Versionierung und Sicherung der Arbeitsergebnisse

In diesem Kapitel wird beschrieben, wie der Lernende sicherstellt, dass die erarbeiteten Ergebnisse während der Probe-IPA sicher gespeichert und jederzeit wieder aufrufbar sind. Die Versionierung soll es ermöglichen, frühere erstellte Versionen der Daten jederzeit wiederherstellen zu können. Die Massnahmen werden hier vom Lernenden aufgeführt.

5.1 Verwendung von Git zu Versionierung

Für die Versionierung von der Probe-IPA wird Git verwendet. Git ist ein Versionierungstool und wird genutzt, um Quellcode zu versionieren und zu beschriften. In der Schule so wie auch in der Firma wurde Git bereits in diversen Projekten verwendet, um den Quellcode übersichtlich zu versionieren und in der Cloud zu sichern. Mit Git kann man sogenannte «Commits» machen, um einen kleinen Teil der Änderungen zu speichern und zu beschriften. Diese Änderungen kann man jederzeit wieder rückgängig machen oder aufrufen, um eine bestimmte Version genauer zu analysieren. Durch diese Commits ist der Quellcode für eine andere Person verständlicher zu lesen.

5.2 Quellcode

Der Quellcode der eingehenden Message-Queue-Nachrichten im Web-GUI wird mit Git verwaltet und ist in einem Repository auf Bitbucket gespeichert. In Abbildung 5.1 ist die Git Commit History des Quellcodes ersichtlich.



Abbildung 5.1: Git Commit History des Quellcodes

5.3 Probe-IPA Dokumentation

Die Probe-IPA-Dokumentation wird mithilfe von \LaTeX geschrieben, wodurch eine Versionierung mit Git auch möglich wird. Die \LaTeX - und alle anderen benötigten Dateien werden auf ein privates Repository in der Cloud geladen. In Abbildung 5.2 ist die Git Commit History der Probe-IPA-Dokumentation ersichtlich.



Abbildung 5.2: Git Commit History der Dokumentation

6 Projektmanagementmethode

In diesem Kapitel ist die Projektmanagement-Methode «IPERKA» beschrieben, die während der Probe-IPA verwendet wird. Es werden die Gründe für diese Projektmanagement-Methode aufgeführt und eine alternative Methode mit Gründen, warum sie nicht verwendet wurde.

6.1 IPERKA

Als Projektmanagement-Methode während der Probe-IPA wird IPERKA verwendet. IPERKA ist eine Vorgehensmethode, die sich gut für Projekte mit überschaubarem Umfang und klar definierte Ziele eignet. Die Methode wurde bereits im ersten Lehrjahr in der Schule behandelt und ist durch das bereits bekannt. Der Name «IPERKA» setzt sich aus den Anfangsbuchstaben der sechs verschiedenen Schritten zusammen, nach denen vorgegangen wird:

I nformieren: Den Auftrag verstehen, eine Vorstellung der Lösung erhalten, fehlende Informationen einholen, ordnen und bewerten

P lanen: Nötige Arbeitsschritte definieren, einen Zeitplan erstellen, Methoden und Arbeitsmittel definieren

E ntcheiden: Verschiedene Lösungsvarianten vergleichen, ausschlaggebende Kriterien definieren, eine Lösungsvariante auswählen

R ealisieren: Arbeit umsetzen, Arbeitsschritte und Ergebnisse dokumentieren, auftretende Probleme behandeln

K ontrollieren: Arbeit testen, Resultate mit den Anforderungen vergleichen, Soll-Ist-Vergleich des Zeitplans, Dokumentation nochmals durchlesen

A uswerten: Rückblick auf das Vorgehen, Arbeitsschritte beurteilen, Selbsteinschätzung vornehmen, mögliche Optimierungen für weitere Projekte definieren

Die Aufteilung der Arbeit in die genannten Schritte unterstützt dabei, die Aufgaben sinnvoll zu strukturieren und systematisch vorzugehen. Ausserdem hilft die bewusste Steuerung des Arbeitsprozesses, die persönlichen Kompetenzen weiterzuentwickeln (vgl. ICT Berufsbildung Bern 2024(Bern 2024)). Die klare Trennung der Schritte stellt sicher, dass der Umfang und die erwarteten Ergebnisse der Aufgabe sich im Verlauf der Arbeit nicht mehr stark ändern, da IPERKA grundsätzlich kein «Rückwärtsgehen» in den Phasen, wie dies aus iterativen Modellen bekannt ist, vorsieht. Ausserdem stellt sie sicher, dass die Realisierung nicht zu schnell in Angriff genommen wird.

6.2 Alternative Methode

Neben IPERKA wurde noch eine andere Vorgehensmethode angeschaut. Diese ist folgend, jeweils mit einer Begründung, wieso IPERKA der Methode vorgezogen wurde, kurz beschrieben.

Scrum ist eine bekannte agile Vorgehensmethode, die heutzutage in der Softwareentwicklung weit verbreitet ist. Die Methode legt den Fokus mehr auf Punkte wie laufende Software, gute Zusammenarbeit, oder flexibles Reagieren auf Veränderungen, anstatt einem strikten Plan zu folgen. Scrum sieht einen iterativen Prozess vor, der laufend optimiert werden soll, und definiert verschiedene Rollen, die jeweils ihre Aufgabe in diesem Scrum-Prozess haben. Ein «agiles» Vorgehen ist in der Softwareentwicklung grundsätzlich sinnvoll, da oft nicht von Beginn her klar ist, wie das Resultat schlussendlich aussehen soll. Da die Probe-IPA schlussendlich aber eine Prüfung ist, sind die Vorgaben und erwarteten Resultate relativ klar. Auch der Umfang der Probe-IPA und das Enddatum sind von Beginn an bekannt und verändern sich nicht während der Arbeit. Ausserdem wird die Aufgabe von nur einer Person bearbeitet, wofür die Abläufe von Scrum nicht optimal geeignet sind. Aus diesen Gründen wird IPERKA gegenüber Scrum vom Lernenden bevorzugt.

7 Arbeitsprotokoll

Datum	06.11.2024
Bearbeitete Arbeitspakete	7.1, 7.2, 7.3, 7.4
Arbeitszeit	8h
Überzeit	0h
Vergleich mit dem Zeitplan	Zeitplan noch nicht fertig
Erfolge und Probleme	Durch die Vorlage des Dokuments und L ^A T _E X konnte ich direkt mit dem ersten Teil der Dokumentation beginnen, was mir viel Zeit erspart hat. Ich habe heute mein Ziel, den ersten Teil grösstenteils abzuschliessen, erreicht und hatte keine grösseren Probleme die aufgetreten sind.
Tagesreflexion	Ich bin heute gut in die Probe-IPA gestartet. Anfangs war ich ein wenig übervordert und wusste nicht wo ich anfangen sollte. Nach der ersten Stunde hat sich das aber wieder gelegt und ich konnte konzentriert an meinem Ziel arbeiten.
In Anspruch genommene Hilfe	Keine

Datum	07.11.2024
Bearbeitete Arbeitspakete	1.1, 1.2, 2.1, 2.2
Arbeitszeit	8h
Überzeit	0h
Vergleich mit dem Zeitplan	Ich konnte alle geplanten Arbeiten von heute Erledigen und hatte auch eine Stunde übrig. Ich habe also bereits mit der Aufgabe 2.3 angefangen.
Erfolge und Probleme	Heute konnte ich mit dem 2. Teil der Dokumentation anfangen. Die Phase Informieren konnte ich Zeitgerecht abschliessen und habe bereits in der Phase Planen die Arbeitspakete und der Zeitplan fertig gestellt. Momentan habe ich mit der Aufgabe 10.1.2 begonnen die für Morgen eingeplant ist.
Tagesreflexion	Ich konnte heute motiviert in den Tag starten, da ich gestern meine Tagesziele erreicht habe. Ich konnte mich nicht wirklich für 10.1.1 und 10.1.1 motivieren, wusste aber, dass ich mich danach mit den Arbeitspaketen und dem Zeitplan beschäftigen konnte. Diese zwei Teile haben mir Spass gemacht, da ich mich danach an dem Zeitplan orientieren kann.
In Anspruch genommene Hilfe	Ich habe mich bei Loris Diana 2 erkundigt, ob ich die Codequalität mit dem Tool, welches mein Projekt nutzt, überprüfen darf, oder ich selbst diese Überprüfung machen muss.

Datum	08.11.2024
Bearbeitete Arbeitspakete	2.3, 2.4
Arbeitszeit	8h
Überzeit	0h
Vergleich mit dem Zeitplan	Ich konnte alle geplanten Arbeitspakete machen oder anfangen.
Erfolge und Probleme	Heute hatte ich Probleme mit der Konzentration. Sie ist schwungartig gekommen und wieder gegangen. Durch das habe ich ein wenig zeit verloren und bin jetzt wieder im Zeitplan. Mein Ziel für heute war eigentlich das Arbeitspaket 2.5 fertig zu haben, sodass ich nächste Woche mit dem Testkonzept starten kann. Ich habe es nicht ganz geschafft. es fehlt aber nicht mehr viel.
Tagesreflexion	Wie bei Erfolge und Probleme schon erwähnt fehlte mir heute ein wenig die Konzentration. Ich glaube es könnte mit der VA zusammenhängen, da ich diese Woche noch daran gearbeitet habe und ich so viel geschrieben haben. Ansonsten kann ich mich nicht beklagen, da ich immer noch ein kleines bisschen im Vorsprung zum Zeitplan bin.
In Anspruch genommene Hilfe	Ich habe mich heute bei Dominic Monzón 2 erkundigt ob für die Filterung alle MQ_IN_STATI gebraucht werden, weil im Quellcode die einen mit «// TODO huberdav: can be removed eventually» beschriftet sind.

Datum	12.11.2024
Bearbeitete Arbeitspakete	2.5
Arbeitszeit	3.5h
Überzeit	-0.5h
Vergleich mit dem Zeitplan	Momentan bin ich auf Kurs mit dem Zeitplan und habe keine Abweichungen.
Erfolge und Probleme	Alle Lösungskonzepte sind fertig und ich konnte mit den Testkonzepten anfangen.
Tagesreflexion	Heute kam ich gut voran und bin immer noch im Zeitplan. Der Vorsprung von letzter Woche ist heute nicht mehr vorhanden, aber ich bin nicht im Verzug. Heute war die Konzentration wieder hoch und ich hoffe, das bleibt auch für die bleibenden Tage.
In Anspruch genommene Hilfe	Ich habe mich bei Loris Diana 2 erkundigt, ob ein Integration-Test für einen Endpoint eine Datenbank braucht die läuft. Er hat mir erklärt, dass es eine braucht aber auch eine In-Memory-Datenbnk auch reicht.

Datum	13.11.2024
Bearbeitete Arbeitspakete	2.6, 3.1, 4.1
Arbeitszeit	8.5h
Überzeit	0.5h
Vergleich mit dem Zeitplan	Durch die nicht verwendeten 3 Stunden von der Entscheidung wurde ich heute bei dem Punkt 4.1 2 Stunden früher als geplant fertig.
Erfolge und Probleme	Heute konnte ich nach vielen Tagen Dokumentation schreiben endlich mit dem programmieren anfangen. Ich habe den Endpunkt bereits fertig und Dokumentiere diesen jetzt.
Tagesreflexion	Mit viel Motivation startete ich heute in den Tag, da ich wusste, dass ich endlich mit dem programmieren anfangen kann. Ich habe das Textkonzept fertig geschrieben und konnte ziemlich schnell die Entscheidung abschliessen und habe so zwei Meilensteine in meiner Probe-IPA erreicht.
In Anspruch genommene Hilfe	Ich hatte Probleme mit einem Bean in Java welches für den MqIn-Service gebraucht wurde. Dieses wurde nicht gefunden und Dominic Monzón 2 hat mir geholfen dieses für der Service zu erstellen. Ausserdem konnte ich den Endpoint nicht ansprechen, da die Url fälschlicherweise falsch war und habe mir Hilfe von Micha Schena geholt (Ein Mitarbeiter im CardX-Team).

Datum	14.11.2024
Bearbeitete Arbeitspakete	4.2
Arbeitszeit	8h
Überzeit	0h
Vergleich mit dem Zeitplan	Momentan bin ich synchron mit dem Zeitplan.
Erfolge und Probleme	Heute konnte ich die Mindestanforderungen abschliessen und mit der Erweiterung weiter machen.
Tagesreflexion	Ich konnte heute wieder mit viel Motivation in den Tag starten, da heute die Implementation im Frontend bevor stand. Ich hatt länger als gedacht, da ich gestern einen Endpoint vergessen habe zu implementieren, welcher ich heute nacharbeiten musste.
In Anspruch genommene Hilfe	Loris Diana 2 hat mir heute erklärt wie die Generierung von den HTTP-Requests im Frontend genau funktioniert.

Datum	15.11.2024
Bearbeitete Arbeitspakete	4.3, 4.5
Arbeitszeit	8h
Überzeit	0h
Vergleich mit dem Zeitplan	Da ich die Implementation von 4.4 10.1.4 pausiert habe bin ich wieder synchron mit dem Zeitplan.
Erfolge und Probleme	Heute hatte ich viele Schwierigkeiten mit der Implementierung der Erweiterung im Frontend. Trotz des Beispiels von «Fehlgeschlagene Zahlung», schaffte ich es nicht, ein Filterergebnis im Frontend anzeigen zu lassen. Ich habe vieles ausprobiert und werde, falls noch Zeit ist, mit einem Fachverantwortlichen das Problem nochmals angehen.
Tagesreflexion	Ich hatte heute viel Motivation, um die Implementierung für den grössten Teil zu ermöglichen. Leider schwand diese Motivation ziemlich schnell während der Implementierung der Erweiterung im Frontend. Anfangs kam ich noch gut voran. Mit der Zeit wusste ich aber nicht, ob diese Implementierung funktionieren wird oder nicht. Da ich für beide Implementationen (das Backend und Frontend der Erweiterung) noch keine Dokumentation geschrieben habe, habe ich mich dazu entschieden, die Implementation zu pausieren und damit anzufangen, sodass ich nicht noch mehr in den Verzug komme.
In Anspruch genommene Hilfe	Dominic Moncón 2 hat mir heute bei einer Frage bezüglich der Filterung beantwortet. Für die Filterung von Nachrichten konnte ich nicht auf die entsprechende Spalte zugreifen, und die Frage war, wie ich jetzt auf sie zugreifen kann. Bei anderen Feldern gibt es eine Variable, welche dies ermöglicht. Oberhalb der Felder steht aber, dass sie generiert wurden, und ich wusste nicht, ob ich sie direkt in dieser Klasse hinzufügen kann oder ob ich anderswo eine kleine Änderung unternehmen muss. Ich konnte schlussendlich einfach diese Variable hinzufügen.

Datum	19.11.2024
Bearbeitete Arbeitspakete	5.1 (angefangen)
Arbeitszeit	3h
Überzeit	-1h
Vergleich mit dem Zeitplan	Im Moment bin ich noch im Zeitplan.
Erfolge und Probleme	Heute konnte ich die Phase Realisieren abschliessen und mit der Implementierung der Tests anfangen.
Tagesreflexion	Ich habe heute noch die Mock Implementation vom MqInService gemacht für das Akzeptanzkriterium. Dies ging ziemlich schnell und war nicht schwer zu implementieren. Ausserdem habe ich den ersten Test heute geschrieben, ohne grosse Probleme. Momentan kommen aber die hinzugefügten Daten in der Datenbank noch nicht im Test an. Dieses Problem werde ich aber morgen in Angriff nehmen.
In Anspruch genommene Hilfe	-

Datum	20.11.2024
Bearbeitete Arbeitspakete	5.1, 5.2 (angefangen)
Arbeitszeit	9h
Überzeit	1h
Vergleich mit dem Zeitplan	Durch die Verzögerung mit der Prüfung von der Codequalität bin ich momentan 4 Stunden im Verzug.
Erfolge und Probleme	Heute konnte ich das Schreiben der Tests abschliessen. Leider hatte ich Probleme bei der Codequalität und bin so heute nicht mehr zum finalisieren der Dokumentation gekommen.
Tagesreflexion	Heute war ein langer Tag durch das aufholen der Stunde von gestern. Ich war motiviert die Tests endlich zu implementieren und freute mich mit der Codequalität anzufangen. Leider war ich damit nicht so schnell fertig wie gedacht und bin jetzt im Verzug. Ich rechne aber damit, dass ich mit der Finalisierung der Dokumentation nicht so lange brauche wie im Zeitplan beschrieben, um so den Verzug wieder gut zu machen.
In Anspruch genommene Hilfe	Dominic Moncón 2 hat mit heute bei Problemen mit dem Build (Codequalität prüfen) geholfen, da es einige Schwierigkeiten gab die ich nicht lösen konnte.

Datum	21.11.2024
Bearbeitete Arbeitspakete	...
Arbeitszeit	...
Überzeit	...
Vergleich mit dem Zeitplan	...
Erfolge und Probleme	...
Tagesreflexion	...
In Anspruch genommene Hilfe	...

Datum	22.11.2024
Bearbeitete Arbeitspakete	...
Arbeitszeit	...
Überzeit	...
Vergleich mit dem Zeitplan	...
Erfolge und Probleme	...
Tagesreflexion	...
In Anspruch genommene Hilfe	...

Teil II

Projekt

8 Kurzfassung

Ausgangssituation CardX ist eine Transaktions-Autorisierungs-Lösung, die bei einigen Banken im Einsatz ist. CardX kommuniziert mit diversen anderen bankspezifischen IT-Systemen, zum Beispiel dem Kernbankensystem oder Service-Büros über Message-Queues. Diese Queues werden asynchron durch CardX befüllt und dabei in den Datenbank-Tabellen MQ_TABLE (eingehende Nachrichten) und MQ_OUT (ausgehende Nachrichten) zwischengespeichert. Diese Tabellen können momentan nur direkt auf der Datenbank eingesehen werden.

Umsetzung Im Frontend wird auf die Seite Business Daten navigiert. Nachdem die Seite erreicht wurde, wird im Hintergrund ein GET-Request an das Backend gesendet. Im Backend sorgt dieser Request dafür, dass die Message-Queues aus der Datenbank hervorgeholt werden mithilfe von Hibernate. Die Daten werden bereits durch die SQL-Query sortiert und gefiltert. Anschliessend werden die Daten ins Frontend geschickt und dort mit einer Tabelle angezeigt. Um zu garantieren, dass der Code fehlerfrei läuft, werden noch Tests im Backend geschrieben.

Ergebnis Die neue Seite Business Daten ist korrekt umgesetzt und beinhaltet keine Fehler. Bei jedem Push wird der Code getestet und auf Styling geschaut.

9 Informieren

Dieses Kapitel zeigt die in der IPERKA-Phase «Informieren» durchgeführten Arbeiten auf. In dieser Phase wird der Einsatzzweck und die Funktionsweise genauer analysiert. Basierend darauf werden Anforderungen an das Projekt definiert.

9.1 Projektumfeld

In diesem Abschnitt wird das Projektumfeld genauer analysiert und unter anderem grafisch dargestellt. Es gilt den groben Ablauf der Aufgabenstellung zu kennen um die Implementation zu vereinfachen.

9.1.1 Einsatzzweck

Momentan sind die Message-Queues nur in der Datenbank ersichtlich. Für den alltäglichen Gebrauch ist das Zugreifen auf die Datenbank umständlich und können Probleme auftreten. Da wenig bis gar keine Sicherheitsmassnahmen bei der Bearbeitung von Elementen in der Datenbank existieren, kann das Unterbrüche und andere Probleme verursachen. Diese Probe-IPA soll das Lesen und Bearbeiten dieser Message-Queues vereinfachen.

9.1.2 Funktion

Die neue Seite Business Daten soll eine Tabelle beinhalten mit den folgenden Spalten.

- MODIFIED_AT: Die letzte Änderung an der Message-QUEUE
- MQ_QUEUE_ID: Message-Queue-ID
- JOB_KEY: Der dazugehörige JOB
- MESSAGE_SHORT / _LONG: Die Nachricht
- MQ_IN_STATUS: Der Verarbeitungs-Status
- MQ_IN_STATUS_STRING: Das Verarbeitungs-Ergebnis
- MQ_IN_STATUS_STRING: Die Anzahl an Verarbeitungsversuche

Diese Daten werden dann durch ein Get-Request vom Backend geholt. Das Ziel ist, dass die Daten im Frontend nur noch angezeigt werden müssen für die Mindestanforderungen. Im Backend wird durch eine SQL-Query dafür gesorgt, dass nur die oben genannten Spalten aus der Datenbank hervorgeholt werden. In der Abbildung 9.1 ist der Ablauf eines solchen GET-Requests ersichtlich.

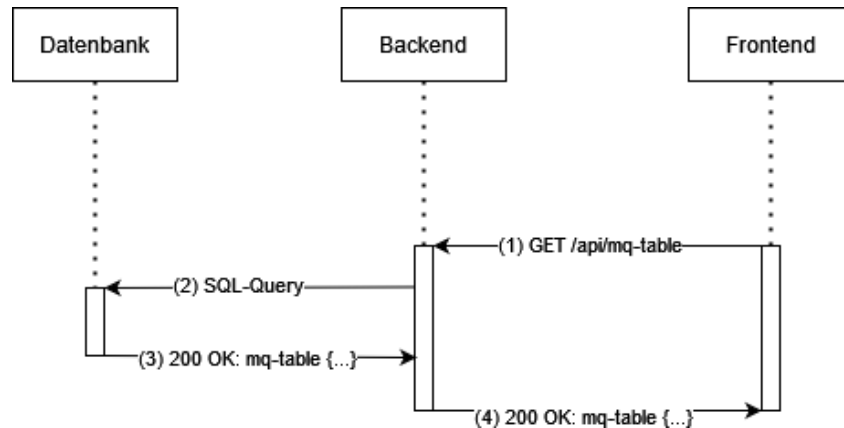


Abbildung 9.1: Ablauf eines GET-Requests für das Erstellen der Tabelle

Folgende Schritte werden im Sequenzdiagramm durchgeführt.

1. Das Frontend sendet einen GET-Request für alle Message-Queues um sie anschliessend in der Tabelle darzustellen.
2. Das Backend empfängt diesen Request und führt eine SQL-Query aus um die entsprechenden Message-Queues zu erhalten. Die SQL-Query filtert bereits alle Spalten heraus die nicht genutzt werden.
3. Die Datenbank schickt eine gefilterte Liste mit Message-Queues zurück an das Backend.
4. Das Backend mappt die entsprechenden Daten in Objekte die das Frontend kennt und schickt es an sie.

Danach kann das Frontend die Daten anzeigen.

9.2 Anforderungen

Auf Basis der unter aufgeführten detaillierten Aufgabestellung und den individuellen Beurteilungskriterien werden die folgenden Anforderungen definiert. Die Anforderungen sind den folgenden drei Teilbereichen entsprechend gruppiert und bezeichnet.

MA... Minimalanforderungen

EP... Erweiterung Pagination

EF... Erweiterung Filter

9.2.1 Minimalanforderungen

Folgende funktionale und nicht funktionale Anforderungen sind hier für die Minimalanforderungen aufgelistet.

Funktionale Anforderungen

Anforderung	Beschreibung
MA1	Die neue Seite ist im Web-GUI auffindbar über den Pfad /admin/mq-table-search und übers Menü unter "Business Daten" → "Queue-Messages".
MA2	Die Tabelle beinhaltet maximal 25 Einträge.
MA3	Alle Einträge sind im Fehlerzustand, d.h. MQ_IN_STATUS == 3.
MA4	Die tabellarische Darstellung von MQ_TABLE beinhaltet folgende Spalten: MODIFIED_AT, MQ_QUEUE_ID, JOB_KEY, MESSAGE_SHORT / _LONG, MQ_IN_STATUS, MQ_IN_STATUS_STRING und MQ_IN_STATUS_STRING.
MA5	Anhand von MESSAGE_IS_SHORT wird entschieden ob MESSAGE_SHORT oder MESSAGE_LONG abgefüllt werden soll.
MA6	Im Web-GUI sollen in der tabellarischen Ansicht nur die ersten 50 Zeichen von MESSAGE_SHORT / _LONG angezeigt werden.
MA7	Der MQ_IN_STATUS wird in textuelle Stati, gemäss ch.ergon.cardx.shared.database.enumeration.MqInStatus, übersetzt.
MA8	Das Kontextmenü beinhaltet eine Aktion für die Anzeige des kompletten Inhalts der Spalte "Nachricht".
MA9	Das Kontextmenü beinhaltet eine Aktion, um einen erneuten Verarbeitungsversuch eines Eintrages anzustossen. D.h. durch das Setzen des MQ_IN_STATUS auf 0.
MA10	Die neuen public-Methoden im Admin-Backend sind mit Unit-Tests bzw. SpringBoot-Tests abgedeckt. Eintrag 4
MA11	Es gibt einen Mock-Datensatz für den neuen MqTable-Rest-Service.

Nicht funktionale Anforderungen

Anforderung	Beschreibung
MA12	Die Seite ist stimmig ins UI eingebaut, bereits existierende UI-Komponenten werden wiederverwendet, oder falls nötig erweitert.
MA13	Die Seite wird in kurzer Zeit geladen, notwendiges Filtering wird auf der Datenbank oder dem Server gemacht.

9.2.2 Erweiterung: Pagination

Folgenden funktionale und nicht funktionale Anforderungen sind hier für die Erweiterung Pagination aufgelistet.

Funktionale Anforderungen

Anforderung	Beschreibung
EP1	Es können mittels Pagination auch mehr als 25 Einträge im Web-GUI angeschaut werden.
EP2	Eine Page beinhaltet maximal 25 Einträge.
EP3	Der Pagination-Mechanismus ist durch Unit-Tests bzw. SpringBoot-Tests abgedeckt.

Nicht funktionale Anforderungen

Anforderung	Beschreibung
EP4	Einzelne Pages werden erst wenn diese gebraucht werden vom Server geladen.
EP5	Die Navigation zwischen den Seiten ist intuitiv und nutzerfreundlich.

9.2.3 Erweiterung: Filter

Folgenden funktionale und nicht funktionale Anforderungen sind hier für die Erweiterung Filter aufgelistet.

Funktionale Anforderungen

Anforderung	Beschreibung
EF1	Es gibt folgende Filter-Kriterien: Filtern nach MQ_IN_STATUS, Filtern mittels Begriffen im Nachrichteninhalt und Filtern nach «Datum von» und «Datum bis»
EF2	Bei Verwendung mehrerer Filter werden diese mittels logischem UND verknüpft.
EF3	Die Filter-Werte können einfach aus dem Web-UI gesetzt werden.
EF4	Die Filterung passiert im Hintergrund, d.h. auf der Datenbank und/oder dem Server.
EF5	Die Einträge sind weiterhin sortiert nach MODIFIED_DATE.
EF6	Alle Filterkriterien und -werte sind in der URL abgebildet.
EF7	Alle Filter sowie ein Beispiel mit einer Kombination von Filtern sind als Unit-Tests bzw. SpringBoot-Tests abgedeckt.

Nicht funktionale Anforderungen

Anforderung	Beschreibung
EF8	Alle Filterkriterien und -werte sind in der URL abgebildet, um das Speichern und Teilen der Filter zu vereinfachen.

10 Planen

Dieses Kapitel zeigt die in der IPERKA-Phase «Planen» durchgeführten Arbeiten auf. In dieser Phase werden basierend auf den Anforderungen Arbeitspakete definiert und in einem Zeitplan auf die zehn Probe-IPA Tage aufgeteilt. Zudem werden Lösungskonzepte für die Umsetzung der Seite Business Daten erarbeitet und ein Testkonzept erstellt.

10.1 Arbeitspakete

Die gesamte Arbeit der Probe-IPA wird in Arbeitspakete aufgeteilt. Die Arbeitspakete bestehen aus einer zugehörigen Nummer, einem Namen, dem geschätzten Aufwand und ein erwartetes Ergebnis. In den geschätzten Aufwand sind Zeitreserven mit einberechnet, sodass unvorhergesehenes kompensiert werden kann. Da der Zeitplan in 2-Stunden-Blöcke aufgeteilt ist, wird der geringste Aufwand 2 Stunden sein und im zweier Takt nach oben gehen.

Die Arbeitspakete sind nach der Projektmanagementmethode IPERKA gegliedert. Probe-IPA-spezifische Arbeiten, wie die Expertenbesuche oder das Erstellen des Anhangs, die ausserhalb der eigentlichen Projekts stehen, werden unter «Rahmenaufgaben» aufgeführt.

10.1.1 Informieren

Folgende Arbeitspakete gehören zu der IPERKA-Phase «Informieren».

Nummer	1.1
Name	Projektumfeld analysieren und beschreiben
Geschätzter Aufwand	2h
Erwartetes Ergebnis	Die Aufgabestellung ist beschrieben und für den Lernenden der Auftrag klar. Der Lernende kennt das Projektumfeld und hat es dokumentiert.

Nummer	1.2
Name	Anforderungen definieren
Geschätzter Aufwand	2h
Erwartetes Ergebnis	Die Anforderungen werden klar definiert und unterteilt in funktionale und nicht funktionale Anforderungen.

10.1.2 Planen

Folgende Arbeitspakete gehören zu der IPERKA-Phase «Planen».

Nummer	2.1
Name	Arbeitspakete definieren
Geschätzter Aufwand	4h
Erwartetes Ergebnis	Die gesamten Arbeitspakete sind definiert und nummeriert nach den Phasen von IPERKA.

Nummer	2.2
Name	Zeitplan erstellen
Geschätzter Aufwand	2h
Erwartetes Ergebnis	Der Zeitplan wird mithilfe der Arbeitspakete erstellt und die bereits erfüllten Aufgaben werden entsprechend markiert.

Nummer	2.3
---------------	------------

Name	Lösungskonzept für die Struktur vom Backend
Geschätzter Aufwand	2h
Erwartetes Ergebnis	Ein Lösungskonzept für die Struktur im Backend wird erarbeitet und dokumentiert.

Nummer	2.4
Name	Lösungskonzept für die Struktur vom Frontend
Geschätzter Aufwand	2h
Erwartetes Ergebnis	Ein Lösungskonzept für die Struktur im Frontend wird erarbeitet und dokumentiert.

Nummer	2.5
Name	Lösungskonzept für die Struktur von einer Erweiterung
Geschätzter Aufwand	4h
Erwartetes Ergebnis	Der Lernende entscheidet sich zwischen einer der beiden Erweiterungen und erarbeitet für dieses ein Lösungskonzept und dokumentiert diese anschliessend.

Nummer	2.6
Name	Testkonzept erstellen
Geschätzter Aufwand	4h
Erwartetes Ergebnis	Ein Testkonzept wird erarbeitet. Die zu schreibenden Tests und Testergebnisse sind definiert.

10.1.3 Entscheiden

Folgende Arbeitspakete gehören zu der IPERKA-Phase «Entscheiden».

Nummer	3.1
Name	Lösungsvarianten evaluieren
Geschätzter Aufwand	4h
Erwartetes Ergebnis	Mögliche Lösungsvarianten wurden evaluiert und die umzusetzende Lösungsvariante ist definiert.

10.1.4 Realisieren

Folgende Arbeitspakete gehören zu der IPERKA-Phase «Realisieren».

Nummer	4.1
Name	Endpoints für Mindestanforderungen erstellen
Geschätzter Aufwand	4h
Erwartetes Ergebnis	Die Endpoints für die Mindestanforderungen werden erstellt, sodass das Frontend alle Daten, die es braucht, und nur die, die es braucht, bekommt.

Nummer	4.2
Name	Seite und Tabelle im Frontend erstellen
Geschätzter Aufwand	4h
Erwartetes Ergebnis	Die Seite ist mit der entsprechenden URL und im Menü unter Business Daten → Queue-Messages (eingehend) erreichbar. Die Tabelle ist stimmig ins UI eingebaut und entspricht der tabellarischen Darstellung.

Nummer	4.3
Name	Endpoints für Erweiterung erstellen
Geschätzter Aufwand	4h
Erwartetes Ergebnis	Die Endpoints für die Erweiterung werden erstellt, sodass das Frontend alle Daten, die es braucht, und nur die, die es braucht, bekommt.

Nummer	4.4
Name	Erweiterung in die Tabelle integrieren
Geschätzter Aufwand	4h
Erwartetes Ergebnis	Die Erweiterung wird im Frontend in die Tabelle integriert. Das Design der Erweiterung muss stimmig zu der Tabelle und der Seite eingebaut werden.

Nummer	4.5
Name	ReleaseNotes schreiben

Geschätzter Aufwand	1h
Erwartetes Ergebnis	Für die neue Tabelle werden ReleaseNotes geschrieben, um die Tabelle und die Erweiterung zu beschreiben und erklären.

10.1.5 Kontrollieren

Folgende Arbeitspakete gehören zu der IPERKA-Phase «Kontrollieren».

Nummer	5.1
Name	Tests
Geschätzter Aufwand	6h
Erwartetes Ergebnis	Das geplante Testkonzept wird Umgesetzt und bei Bedarf ergänzt.

Nummer	5.2
Name	Codequalität prüfen
Geschätzter Aufwand	2h
Erwartetes Ergebnis	Die Codequalität wird mithilfe von der Jenkins Pipeline überprüft und bei Bedarf überarbeitet.

Nummer	5.3
Name	Dokumentation finalisieren
Geschätzter Aufwand	8h
Erwartetes Ergebnis	Die Dokumentation ist nachvollziehbar und verständlich. Das Dokument wird nach Schreibfehlern durchsucht und verbessert, sodass zu diesem Zeitpunkt fast bis gar keine mehr übrig sind. Die Struktur ist einheitlich und Unschönheiten wurden behoben.

10.1.6 Auswerten

Folgende Arbeitspakete gehören zu der IPERKA-Phase «Auswerten».

Nummer	6.1
---------------	------------

Name	Kurzfassung schreiben
Geschätzter Aufwand	2h
Erwartetes Ergebnis	Das Projekt wird in einer Kurzfassung zusammengefasst.

Nummer	6.2
Name	Reflexion schreiben
Geschätzter Aufwand	2h
Erwartetes Ergebnis	Das Projekt wird vom Lernenden reflektiert und dokumentiert.

10.1.7 Rahmenaufgaben

Folgende Arbeitspakete gehören zu den Rahmenaufgaben.

Nummer	7.1
Name	Projektstruktur aufsetzen
Geschätzter Aufwand	2h
Erwartetes Ergebnis	Aufbau der Gerüstes des L ^A T _E XBerichtes, welches die Titelseite, ein Glossar, Das Quellenverzeichnis und ein Abbildungsverzeichnis beinhaltet. Ein Git Repository wird für die Dokumentation aufgesetzt.

Nummer	7.2
Name	Aufgabenstellung und Rahmenbedingungen beschreiben
Geschätzter Aufwand	2h
Erwartetes Ergebnis	Die Aufgabenstellung und Rahmenbedingungen hat der Lernende verstanden und beschreibt sie nochmals um dies zu bestätigen.

Nummer	7.3
Name	Projektmanagementmethode definieren
Geschätzter Aufwand	2h

Erwartetes Ergebnis	Eine Projektmanagementmethode wird definiert und durch einer anderen wird erleutert warum sie gewählt wird.
----------------------------	---

Nummer	7.4
Name	Expertenbesuche
Geschätzter Aufwand	2h
Erwartetes Ergebnis	Die Expertenbesuche werden erfolgreich geplant und durchgeführt. Wichtige Informationen bezüglich der Besuche werden an passenden Plätzen erwähnt und dokumentiert.

Nummer	7.5
Name	Anhang erstellen
Geschätzter Aufwand	4h
Erwartetes Ergebnis	Ein Anhang mit allen gemachten Änderungen am Quellcode ist dokumentiert und klar von dem unveränderten Code unterscheidbar.

10.2 Lösungskonzept für die Struktur vom Backend

In diesem Abschnitt wird das Lösungskonzept vom Backend für einen Teil der unter 9.2.1 definierten Anforderungen beschrieben.

10.2.1 Erstellung der Endpunkte

Im Backend existieren noch keine Endpunkte für die Message-Queues, um sie im Frontend darzustellen. Deshalb müssen diese neu implementiert werden. Die Endpoints werden in einer Resource-Klasse erstellt und rufen Funktionen von einer Service-Klasse auf. Die Service-Klasse wird von einem Interface implementiert und die Standardfunktionen, wie «getAll» oder «update», sind so bereits im Interface definiert und müssen nur noch in der Service-Klasse implementiert werden. Der Grund für das Interface ist, dass es für die Service-Klasse und die Mock-Service-Klasse genutzt werden kann und beide Klassen die gleichen Funktionen haben, aber mit einer unterschiedlichen Implementierung. Der Zugriff auf die Datenbank erfolgt in einer DAO-Klasse. In dieser Klasse werden mithilfe von Funktionen die Daten von der Datenbank hervorgeholt und bereitgestellt. Diese Funktionen können anschließend von der Service-Klasse aufgerufen und werden von einem DAO-Objekt zu einem DTO-Objekt gemappt, um die Daten in ein passendes Objekt zu füllen, welches anschließend wieder an das Frontend gesendet wird.

10.2.1.1 DAO-Klassen

Eine DAO-Klasse (Data Access Object) wird verwendet um die CRUD-Operationen (Create, Read, Update, Delete) bereitzustellen, die benötigt werden, um Daten zu verwalten. Sie abstrahiert die Verbindung zur Datenbank, sodass der Code ausserhalb der DAO-Klasse unabhängig von der Datenzugriffstechnologie, in diesem Fall Hibernate, bleibt.

10.2.1.2 Klassendiagramm für das Backend

Hier wird ein Klassendiagramm dargestellt, mit den verschiedenen Klassen und Beziehungen, um das Verständnis der Erstellung der Endpunkte zu vereinfachen. Diese Version muss nicht der Implementierung entsprechen. Es kann sein dass während der Implementierung Veränderungen auftreten die nicht in diesem Diagramm dargestellt werden.

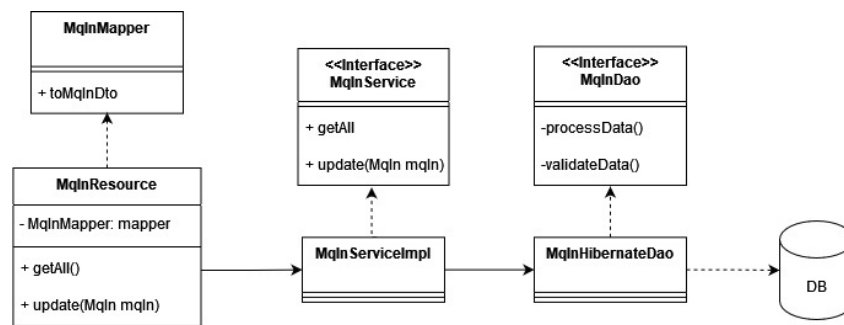


Abbildung 10.1: Klassendiagramm für das Backend

Der Klassenname ist immer oben in der Box in Bold geschrieben. Die Verbindungen zeigen an was passiert, wenn ein Endpoint aufgerufen wird. Bei den Interfaces und den dazugehörigen Implementationen, ist jeweils nur das Interface beschrieben. Die gestrichelten Verbindungen deuten darauf hin, dass sie diese Klasse implementieren oder sie gebrauch von einer Funktion in der entsprechenden Klasse machen.

10.2.2 Verwendete Technologien

Um dieses Konzept umzusetzen werden folgende Technologien verwendet:

Java ist eine bekannte Programmiersprache und wird in diesem Projekt genutzt, um das Backend zu implementieren.

Hibernate ist ein Framework für objekt-relationale Abbildung (ORM) ind java, welcher ermöglicht , Java-Objekte direkt mit relationalen Datenbanken zu verknüpfen.

Springboot ist ein Java-Framework, das auf dem Spring-Framework basiert. Es wurde entwickelt, um die Konfiguration und den Aufbau von Anwendungen zu beschleunigen, indem es automatisierte Konfigurationen und eingebettete Server bietet.

10.3 Lösungskonzept für die Struktur vom Frontend

In diesem Abschnitt wir das Lösungskonzept vom Frontende für einen Teil der unter 9.2.1 definierten Anforderungen beschrieben.

10.3.1 Bestehende Implementationen

Im Frontend besteht bereits vieles. Ein Menü muss nicht mehr erstellt werden. Die Seite kann von anderen Implementationen kopiert werden und passend zu den Minimalanforderungen gemacht werden. Die Darstellung einer Tabelle im Frontend existiert bereits und muss dadurch nicht erneut implementiert werden. Die Tabelle muss jedoch noch angepasst werden, um die richtigen Spalten anzuzeigen. Durch die

bereits existierende Darstellung wird die Tabelle stimmig in das UI eingebaut und wird in etwa gleich aussehen wie die anderen Tabellen im Web-GUI.

10.3.2 Neue Implementationen

Für das Anzeigen der gesamten Nachricht in der Spalte MESSAGE_SHORT / _LONG wird ein Knopf im Kontextmenü verwendet. Wenn die gesamte Nachricht angezeigt wird, wird dieser Knopf durch einen anderen ersetzt, welcher die Nachricht wieder auf 50 Zeichen begrenzt.

Ausserdem beinhaltet das Kontextmenü einen weiteren Knopf um bei einem Eintrag einen neuen Verarbeitungsversuch auszulösen. Dieser Knopf wird bei Einträgen, die bereits an einem Verarbeitungsversuch sind, deaktiviert oder ausgeblendet. Dieser Knopf löst einen Request in das Backend aus, um der neue Verarbeitungsversuch in der Datenbank zu speichern und ihn auszulösen.

10.3.3 Verwendete Technologien

Um dieses Konzept umzusetzen werden folgende Technologien verwendet:

TypeScript ist eine Programmiersprache die verwendet wird um die Funktionalitäten für die Tabelle zu implementieren, wie zum Beispiel werden die Requests, an das Backend, mit TypeScript geschrieben.

Angular ist ein Framework für Single Page Applications (SPAs), bei denen Inhalte dynamisch nachgeladen werden können, ohne dass die Seite neu geladen werden muss. Ausserdem eignet es sich gut um moderne, skalierbare und performante Webanwendungen zu erstellen.

HTML (HyperText Markup Language) wird verwendet um die Struktur und den Inhalt vom Webdokument zu beschreiben. HTML verwendet sogenannte «Tags» die es ermöglichen Text, Bilder und unter anderem auch Tabellen auf der Webseite anzuzeigen.

CSS (Cascading Style Sheets) ist eine Stylesheet-Sprache, die da Design und Layout von HTML-Dokumenten festlegt.

10.4 Entscheidung der Erweiterung

Dieser Absatz erklärt den Grund warum sich der Lernende für eine Erweiterung entschieden hat.

10.4.1 Pagination

Pagination ist etwas Neues für den Lernenden. Er hat es schon öfters gesehen auf anderen Webseiten, hat sich aber noch nie mit der Implementierung auseinandergesetzt. Das Wechseln zwischen mehreren Blättern ermöglicht einen klaren Überblick über die Seite, da die Tabelle durch die Aufteilung eine begrenzte Anzahl Elemente

enthält und die Seite durch das nicht überfüllt wirkt. Die Implementierung existiert bereits bei anderen Tabellen und wäre daher nicht allzu schwer, um sie in die Seite einzubauen.

10.4.2 Filter

Die Implementation für den Filter ist einfacher zu verstehen weil, es Filter auf fast jeder Webseite gibt. ob versteckt oder sichtbar, kann man sich gut vorstellen wie das im Hintergrund ablaufen kann. Es gibt mehr Anforderungen für die Erweiterung Filter, aber sie sind einfacher als die bei dem Paginator. Es gibt nur drei Filter-Kriterien und bei mehreren aktiven Filter werden sie mit einem «UND» verknüpft.

10.4.3 Entscheid

Die Anforderungen von Pagination sind zwar weniger, aber da sich der Lernende noch nie mit Pagination auseinandergesetzt hat erschwert es ihm die Implementation und es könnten Fehler auftreten die viel Zeit kosten. Da die Probe-IPA aber nur 10 Tage zu Verfügung stellt und die Erweiterung Filter schneller gehen könnte, kann der Lernende anschliessend mehr Zeit in die Dokumentation von der Erweiterung investieren, um am Ende ein fertiges Produkt zu präsentieren.

Ausgewählte Erweiterung: Filter

10.5 Lösungskonzept für die Struktur vom Filter

In diesem Abschnitt wird das Lösungskonzept von der Erweiterung Filter der unter 9.2.2 definierten Anforderungen beschrieben.

10.5.1 Bestehende Implementation

Im Web-GUI existiert bereits ein Filter unter «Business Daten» → «Fehlgeschlagene Zahlungen». Dieser Filter beinhaltet, wie auch bei der Erweiterung gefordert, einen «Datum von / Datum bis» Filter, welcher kopiert und angepasst werden kann. Für den neuen Endpoint kann auch der bestehende als Beispiel verwendet werden, um die Implementierung zu erleichtern.

Die Filterkriterien und -werte werden auch bereits bei dem Filter auf der Seite «Fehlgeschlagene Zahlungen» in der URL abgebildet. Dadurch kann für die funktionale Anforderung EF6 dies als Beispiel genutzt werden oder sogar teilweise kopiert werden.

10.5.2 Neue Implementationen

Für die zwei anderen Filterkriterien (filtern nach MQ_IN_STATUS und Filtern mittels Begriffen im Nachrichteninhalt) gibt es noch keine Implementierung in diesem Projekt, weshalb sie selbst erstellt werden müssen.

10.5.2.1 MQ_IN_STATUS

Für diesen Filter wird eine Auswahlliste erstellt mit der folgenden Auswahl:

NEW Eine neue Nachricht, die bereit für die Bearbeitung ist, für den entsprechenden Job.

STOPPED Das Bearbeiten der Nachricht wurde gestoppt und wird später weiter bearbeitet.

WAIT Die Nachricht wurde auf warten gestellt und wird später bearbeitet.

ERROR Die Nachricht kann nicht wegen eines Fehlers bearbeitet werden und muss manuell bearbeitet werden.

CANCELLED Der Status kann manuell gesetzt werden zu dem Status STOPPED, um sie vom Löschjob löschen zu lassen.

NOTICED Der Status kann manuell gesetzt werden zu dem Status ERROR, um sie vom Löschjob löschen zu lassen.

PROCESSED Die Nachricht wurde erfolgreich bearbeitet.

OUTDATED Die Nachricht war veraltet, was bedeutet, dass der Job bereits eine neuere Nachricht bearbeitet hat. (z. B. eine Nachricht mit einer höheren Sequenz-Nummer)

Der ausgewählte Status wird ins Backend geschickt. Die Funktion von diesem Endpoint, welcher den Request bekommt, besteht daraus einen Anfrage an die Datenbank zu machen. Die Filterung, wie in den Anforderungen 9.2.2 beschrieben, passiert darauf hin direkt auf der Datenbank um die Performance so wenig wie möglich zu beeinflussen. Würde die Filterung im Backend geschehen, würde das die Performance ein wenig beeinflussen, weshalb es gleich auf der Datenbank gemacht wird. Dies gilt auch für die beiden anderen Filter.

10.5.3 Begriffe im Nachrichten Inhalt Filtern

Dieser Filter wird mithilfe von einem Textfeld erstellt. In dieses Textfeld können Begriffe hineingeschrieben und durch einen Knopf gefiltert werden. Hier wird auch wieder ein Request an das Backend gesendet, um den Nachrichteninhalt von den Elementen in der Datenbank-Tabelle MQ_TABLE nach diesen Begriffen zu filtern und anschliessend das Ergebnis wieder zurückzusenden an das Frontend.

10.5.4 Sequenzdiagramm für die Erweiterung Filter

In diesem Diagramm wird der Ablauf von einem Filter-Request an das Backend dargestellt. Wegen Zeitgründen werden nicht alle Fälle dargestellt. Sie beruht nur auf einem, zwei und drei Filtern, da alle anderen Möglichkeiten einen ähnlichen Ablauf haben wie die hier dargestellten drei Abläufe.

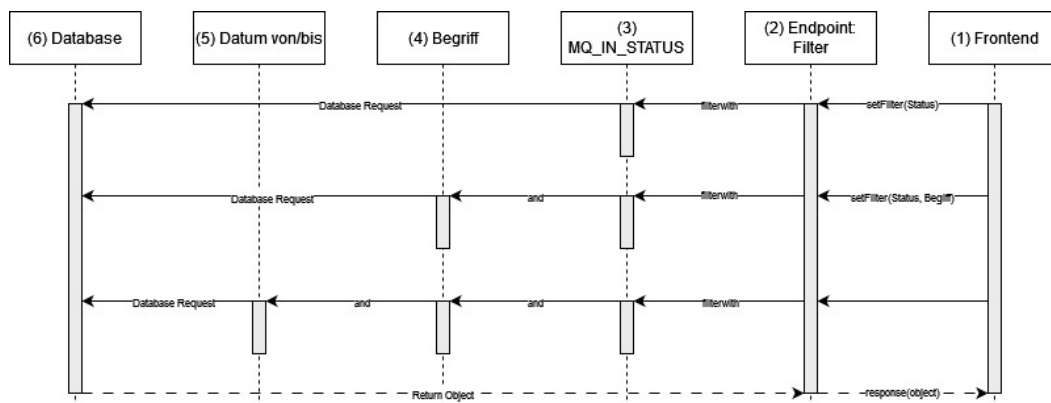


Abbildung 10.2: Ablauf eines Filter-Request

Folgende Schritte werden im Sequenzdiagramm durchgeführt.

1. Im Frontend wird durch das auswählen eines Filters ein Request an das Backend gesendet.
2. Im Backend wird dieser Request angenommen und es wird geprüft welcher Filter genutzt werden sollte.
3. Im Falle von einem Filter wird nichts gross angepasst und eine Anfrage an die Datenbank wird gemacht.
4. Falls zwei Filter ausgewählt werden, wird eine SQL-Query generiert die ein AND zwischen beiden Filtern beinhaltet um nach beidem gleichzeitig zu filtern. Anschliessend wird eine Anfrage an die Datenbank gemacht.
5. Bei allen drei Filter wird, ähnlich wie bei zwei Filter, ein AND zwischen allen drei Filter hinzugefügt und eine Anfrage an das Backend gemacht.
6. Das erhaltene Resultat von der Datenbank wird dann gemapped und zurück an das Frontend gesendet.

10.6 Testkonzept

Um sicherzustellen, dass die Funktionalität der Implementation fehlerfrei funktioniert und alle Anforderungen richtig eingebunden wurden, wird ein Testkonzept erstellt. Darin befindet sich eine kleine Beschreibung, die Art des Tests, die Vorbedingungen, eine Konfiguration, der Ablauf und das erwartete Resultat.

10.6.1 Benötigte Testmittel

Um die Tests zu implementieren werden verschiedene Tools und Technologien verwendet. Diese werden hier aufgeführt:

IntelliJ (IDE) ¹ ist die Entwicklungsumgebung in der die Tests geschrieben werden.

Postman ² wird für das manuelle Testen von den Endpoints verwendet.

Mockito ³ wird für das Mocken von Objekten und Klassen verwendet. Mocking ersetzt Objekte und Klassen durch simulierte Versionen um das erwartete Verhalten nachzuahmen.

10.6.2 Automatisierte Tests

Automatisierte Tests werden nur für die Funktionen im Backend geschrieben. Dies aus dem Grund, weil im Projekt CardX keine Frontend-Tests existieren und diese Implementationen gleich aufgebaut sein sollen wie die anderen Implementationen im Frontend. Sie werden mithilfe von Unit- und Integration-Tests implementiert.

Die Unit-Tests werden für einzelne Funktionen, wie zum Beispiel einen Mapper, verwendet. Sie brauchen keine anderen Systeme und können ohne weiteres ausgeführt werden.

Die Integration-Tests werden oft für Datenbank-Abfragen verwendet, um zu prüfen, ob die Anfrage und das erhaltene Objekt korrekt sind. Diese Art von Testing testet die Zusammenarbeit von verschiedenen Komponenten in einem System, in diesem Fall das Backend und die Datenbank.

10.7 Manuelle Tests

Die manuellen Tests werden für das Backend, mit Postman, und das Frontend durchgeführt. Die neuen Komponenten werden vom Lernenden selbst in einer lokalen Umgebung getestet. Sie werden während der Implementation durchgeführt, um nicht später auf Fehler zu treffen und die Implementation zu vereinfachen.

¹<https://www.jetbrains.com/de-de/idea/>

²<https://www.postman.com/>

³<https://site.mockito.org/>

10.8 Testfälle

Alle Automatischen Tests werden auf 25 Einträge limitiert. Sie haben alle den Fehlerzustand MQ_IN_STATUS und sind absteigend sortiert nach MODIFIED_AT. Die erhaltenen Einträge beinhalten nur die Spalten die vorgegeben sind.

10.8.1 Testfälle der Mindestanforderungen

Testfall	M1
Beschreibung	Datenbank-Abfrage für die Erstellung von der Tabelle
Art	Integration-Test
Vorbedingungen	Das Backend läuft und eine passende Datenbank existiert.
Konfiguration	<ul style="list-style-type: none">– Die Daten für den Aufruf werden erstellt.– Die erwarteten Daten werden erstellt.
Ablauf	<ol style="list-style-type: none">1. Die Datenbank-Abfrage wird durchgeführt.2. Die erhaltenen Daten werden überprüft mit den bereits erstellten Daten.
Erwartetes Ergebnis	Es gibt keine Unterschiede zwischen den Daten aus der Datenbank und den erwarteten Daten.

Testfall	M2
Beschreibung	Datenbank-Abfrage findet keine passende Abfrage.
Art	Integration-Test
Vorbedingungen	Das Backend läuft und eine passende Datenbank existiert.
Konfiguration	<ul style="list-style-type: none">– Der erwartete Fehlerzustand wird erstellt.
Ablauf	<ol style="list-style-type: none">1. Die Datenbank-Abfrage wird durchgeführt.2. Der Code kann mit dem Sonderfall umgehen und schreibt eine Fehlernachricht.

Erwartetes Ergebnis	Das Programm wirft keinen Fehler und sendet stattdessen eine Fehlermeldung.
Testfall	M3
Beschreibung	Datenbank ist nicht erreichbar.
Art	Integration-Test
Vorbedingungen	Das Backend läuft und die Datenbank existiert nicht.
Konfiguration	<ul style="list-style-type: none"> – Der erwartete Fehlerzustand wird erstellt.
Ablauf	<ol style="list-style-type: none"> 1. Die Datenbank-Abfrage wird durchgeführt. 2. Der Code kann mit dem Sonderfall umgehen und schreibt eine Fehlermeldung.
Erwartetes Ergebnis	Das Programm wirft keinen Fehler und sendet stattdessen eine Fehlermeldung.

Testfall	M4
Beschreibung	Mapping von DAO zu DTO
Art	Unit-Test
Vorbedingungen	Das Backend läuft.
Konfiguration	<ul style="list-style-type: none"> – Ein DAO-Objekt ist erstellt und gefüllt mit Daten. – Ein DTO-Objekt ist erstellt und gefüllt mit den gemappten Daten.
Ablauf	<ol style="list-style-type: none"> 1. Die Funktion <code>mapDaoToDto()</code> ausführen mit dem DAO Objekt als Parameter. 2. Der erhaltene Wert mit dem DTO-Objekt vergleichen.
Erwartetes Ergebnis	Es gibt keine Unterschiede zwischen dem generierten DTO- und dem bereits existierendem DTO-Objekt.

10.8.2 Testfälle der Erweiterung Filter

Testfall	M5
Beschreibung	Datenbank-Abfrage für das Filtern von einem Filter
Art	Integration-Test
Vorbedingungen	Das Backend läuft und eine passende Datenbank existiert.
Konfiguration	<ul style="list-style-type: none">– Die Daten für den Aufruf werden erstellt.– Der Filter wird gesetzt.– Die erwarteten Daten werden erstellt.
Ablauf	<ol style="list-style-type: none">1. Die Datenbank-Abfrage wird durchgeführt.2. Die erhaltenen Daten werden überprüft mit den bereits erstellten Daten.
Erwartetes Ergebnis	Es gibt keine Unterschiede zwischen den gefilterten Daten aus der Datenbank und den erwarteten Daten.

Testfall	M6
Beschreibung	Datenbank-Abfrage für das Filtern von mehreren Filter
Art	Integration-Test
Vorbedingungen	Das Backend läuft und eine passende Datenbank existiert.
Konfiguration	<ul style="list-style-type: none">– Die Daten für den Aufruf werden erstellt.– Die Filter werden gesetzt.– Die erwarteten Daten werden erstellt.
Ablauf	<ol style="list-style-type: none">1. Die Datenbank-Abfrage wird durchgeführt.2. Die erhaltenen Daten werden überprüft mit den bereits erstellten Daten.

Erwartetes Ergebnis	Es gibt keine Unterschiede zwischen den gefilterten Daten aus der Datenbank und den erwarteten Daten.
----------------------------	---

Testfall	M7
Beschreibung	Generieren der SQL-Query für einen Filter
Art	Unit-Test
Vorbedingungen	Das Backend läuft.
Konfiguration	<ul style="list-style-type: none"> – Der Filter wird gesetzt. – Das erwartete Ergebnis wird erstellt.
Ablauf	<ol style="list-style-type: none"> 1. Die Funktion wird aufgerufen. 2. Die erhaltene SQL-Query wird überprüft.
Erwartetes Ergebnis	Die erhaltene SQL-Query stimmt mit dem erwarteten Ergebnis überein.

Testfall	M8
Beschreibung	Generieren der SQL-Query für mehrere Filter
Art	Unit-Test
Vorbedingungen	Das Backend läuft.
Konfiguration	<ul style="list-style-type: none"> – Die Filter werden gesetzt. – Das erwartete Ergebnis wird erstellt.
Ablauf	<ol style="list-style-type: none"> 1. Die Funktion wird aufgerufen. 2. Die erhaltene SQL-Query wird überprüft.
Erwartetes Ergebnis	Die erhaltene SQL-Query stimmt mit dem erwarteten Ergebnis überein.

11 Entscheiden

Dieses Kapitel zeigt die in der IPERKA-Phase «Entscheiden» durchgeführten Arbeiten auf. In den Akzeptanzkriterien der Minimalanforderungen wird angegeben, dass die Seite in kurzer Zeit laden soll, weshalb das Filtern auf dem Server oder auf der Datenbank gemacht werden soll. Hier werden beide Lösungsvarianten zueinander verglichen, ihre Vor- und Nachteile aufgelistet und es wird entschieden, welche der beiden implementiert wird.

11.1 Filterung der Daten auf dem Server

11.1.1 Vorteile

Flexibilität und Anpassbarkeit Die Filterlogik kann leicht im Backend angepasst werden bei Bedarf, ohne dass möglicherweise Änderungen an der Datenbank erforderlich sind.

Unabhängigkeit von der Datenbank Durch das Filtern der Daten ausserhalb der Datenbank ist man nicht von Funktionen oder Einschränkungen abhängig und kann bei jeder Datenbankart durchgeführt werden.

Mehr Kontrolle über die Logik Im Backend können zusätzliche Filterlogiken implementiert werden, die in SQL nur schwer oder ineffizient umgesetzt werden können.

11.1.2 Nachteile

Leistungsprobleme bei vielen Daten Bei vielen Daten, die im Backend gefiltert werden müssen, kann dies die Netzwerkauslastung und die Speichernutzung negativ beeinflussen.

Langsamere Antwortzeiten Da die Filterung erst nach der Übertragung erfolgt, kann es zu langsameren Antwortzeiten kommen.

Zusätzliche Verarbeitungsschritte Datenbanken sind oft für die Datenverarbeitung optimiert, während das Backend zusätzliche Verarbeitungsschritte benötigt, was längere Antwortzeiten verursachen kann.

11.2 Filterung der Daten auf der Datenbank

11.2.1 Vorteile

Effizienteres Filtering Da Datenbanken optimiert für Filteroperationen sind, werden nur relevante Daten an das Backend übertragen, was Zeit und die Netzwerklast reduziert.

Reduzierter Speicherbedarf im Backend Durch dass die Daten direkt in der Datenbank gefiltert werden, muss das Backend weniger Arbeitsspeicher verwenden.

Minimierung der Datenübertragung Bei vielen Daten oder langsamen Netzwerkverbindungen kann die Übertragung von gefilterten Daten die Performance erhöhen.

11.2.2 Nachteile

Komplexität von Abfragen Komplexe Filterlogiken können Abfragen komplexer und weniger wartbar machen.

SQL-Kenntnisse erforderlich Komplexe Abfragen brauchen ein gutes Fachwissen, um sie zu schreiben oder anzupassen.

Datenbankbelastung Bei vielen komplexen Filteroperationen oder Abfragen gleichzeitig kann das die Datenbank zusätzlich belasten und die Antwortzeit verlängern.

11.3 Entscheidung

Die Variante mit dem Filtern im Backend hat einige Vorteile wie die Unabhängigkeit von der Datenbank, wodurch man flexibler arbeiten kann. Jedoch ist in den Akzeptanzkriterien die Performance wichtig. Die Datenbank ist für das Filtern von Daten optimiert und ist so im Vorteil. Da die Daten auf 25 Einträge limitiert sein werden, trifft auch der Nachteil von zu grossen Datenmengen nicht in kraft. Die Filteroperationen sind auch nicht zu komplex und die Performance wird durch das nicht beeinträchtigt. Für die Implementierung wird in diesem Fall die Datenbankebene gewählt.

12 Realisieren

Dieses Kapitel zeigt die in der IPERKA-Phase «Realisieren» durchgeführten Arbeiten auf. In dieser Phase wird beschrieben wie der Lernende die Aufgabe umgesetzt hat und spricht Probleme bei der Umsetzung an.

12.1 Endpoints für die Mindestanforderungen erstellen

Die Reihenfolge, welche für diesen Abschnitt folgt, ist die Reihenfolge in der implementiert wurde.

12.1.1 MqInDTO

Die Implementierung wurde mit der Erstellung von der MqInDto.java Klasse gestartet. Diese Klasse wird öfter in anderen Klassen verwendet und ist aus diesem Grund ein guter Startpunkt. Sie wurde mithilfe der MqOutDto.java Klasse als Beispiel erstellt, um die Konsistenz zwischen den verschiedenen DTO-Klassen beizubehalten. Sie beinhaltet mehrere Annotationen, um den Code kurz zu halten und Zeit bei der Implementation zu sparen:

@Data ¹ generiert Getter, Setter und mehr .

@SuperBuilder ² erstellt ein Builder, welcher auch Felder von einer Superklasse verwenden kann.

@NoArgsConstructor ³ generiert einen Konstruktor ohne Parameter.

@Schema ⁴ für die Kontrolle von spezifischen Definitionen wie Beschreibung oder Beispiele.

@NotNull ⁵ stellt sich, dass das Feld nicht «null» ist.

Anschliessend wurden die Felder erstellt, mit den oben entsprechenden genannten Annotationen, die für die Tabelle im Frontend genutzt werden.

¹<https://projectlombok.org/features/Data>

²<https://projectlombok.org/features/experimental/SuperBuilder>

³<https://projectlombok.org/features/constructor>

⁴<https://www.baeldung.com/swagger-parameter-vs-schema>

⁵<https://www.baeldung.com/java-notnull-method-parameter>

12.1.2 MqInMapper

Die Klasse MqInMapper.java hat die Annotation @Component, sodass Springboot diese Klasse instanziiert und sie mit allen angegebenen Abhängigkeiten injiziert kann.

Zwei Funktionen namens toMqInDto() und listToMqInDto() wurden in dieser Klasse erstellt. Die erste Funktion macht ein Mapping von MqTablePC (die ursprüngliche MqInPC-Klasse) zu MqInDto. Sie verwendet den Builder von der zuvor genutzten Annotation @SuperBuilder. Die zweite Funktion ruft die erste mithilfe von einem Stream auf. Der Stream wird verwendet, um jedes Element der Liste mit der Funktion toMqInDto aufzurufen. Dies wird mit .map() gemacht. Der Stream ermöglicht es, diese Aufgabe in einer kurzen Zeile zu schreiben, anstatt mit einem Loop jedes einzelne Element der Liste hervorzuholen, zu mappen und anschliessend in einer zweiten Liste zu speichern. Die Funktion hat durch das nur insgesamt 3 Zeilen und vereinfacht das Lesen.

```
public List<MqInDto> listToMqInDto(List<MqTablePC> listOfMqTablePCs) {  
    return listOfMqTablePCs.stream().map(this::toMqInDto).toList();  
}
```

12.1.3 MqInService

Um die Struktur von einer Service vorzugeben, wurde ein Interface mit dem Namen MqInService erstellt. Dieses Interface wird später für die Klasse MqInServiceImpl.java und MqInServiceMockImpl.java verwendet und beinhaltet im Moment zwei Funktionen für das Hervorholen von allen MqTables in der Datenbank und das Ausführen von einem neuen Verarbeitungsversuch.

12.1.4 MqInServiceImpl

MqInServiceImpl wird vom Interface MqInService implementiert. Sie wird mit der Annotation @Service für Springboot als Service deklariert. Die Klasse enthält die vom Interface vorgegebene Funktion getMqTables(). Sie ruft eine Funktion von der Klasse MqTableDao (die ursprüngliche MqInDao-Klasse) auf die alle MqTable Einträge mit dem Status «Error» von der Datenbank hervorholt und sie auf 25 Einträge limitiert. Ein ähnliches Vorgehen hat die Funktion executeNewProcessingAttempt() die den Status vom MqIn auf NEW setzt.

In dieser Klasse wurde auch ein Konstruktor erstellt. Bei dieser Implementierung trat ein Problem auf, welches die Applikation nicht starten liess. Bei dem Parameter MqTableDao erschien die Meldung «Could not autowire. No beans of 'MqTableDao' type found.». Das Problem war, dass für die Klasse MqTableDao noch kein Bean erstellt wurde, obwohl die Klasse schon vorher existierte. Das Bean wurde anschliessend in der Klasse WebBackendAdminSpringConfiguration.java erstellt.

12.1.5 MqTableDao

Diese Klasse existierte bereits und war ursprünglich als MqInDao geplant. Durch das mussten nur die Funktionen hinzugefügt werden. Da es ein Interface ist, wurden nur die Namen und die benötigten Parameter hinzugefügt. Es wurden noch keine

Bodys implementiert. Ausserdem wurden bei beiden Funktionen ein Kommentar erstellt. Der Kommentar beinhaltet eine kurze Beschreibung der Funktion und was sie zurückgibt. Mithilfe dieses Kommentars kann man jetzt über die Funktion drüberfahren und die Beschreibung wird als kurze Erklärung angezeigt.

12.1.6 MqTableHibernateDao

In der Klasse MqTableHibernateDao wird das Interface MqTableDao implementiert und muss durch das auch die neu erstellte Funktion implementieren.

Die Funktion nutzt einen CriteriaBuilder, um eine SQL-Query zu erstellen. Mit dieser Klasse konnte die Query so generiert werden, dass sie nach Einträgen filtert, die einen MQ_IN_STATUS von ERROR, als die Nummer Drei, haben. Die Query wird mithilfe von der Klasse CriteriaQuery bearbeitet und mit der Klasse Root können die einzelnen Spalten hervorgeholt werden, um sie zu vergleichen.

Um die Einträge auf 25 zu limitieren, musste noch die Klasse TypedQuery verwendet werden, welche dies ermöglichte.

```
@Override
public List<MqTablePC> findAllWithStatusErrorLimitedTo25() {
    CriteriaBuilder cb = getSession().getCriteriaBuilder();
    CriteriaQuery<MqTablePC> query = cb.createQuery(MqTablePC.class);
    Root<MqTablePC> mqTable = query.from(MqTablePC.class);

    query.where(
        cb.equal(mqTable.get(FN_MQ_IN_STATUS), ERROR)
    ).orderBy(cb.desc(mqTable.get(FN_MODIFIED_AT)));

    TypedQuery<MqTablePC> typedQuery = getSession().createQuery(query);
    typedQuery.setMaxResults(25);

    return typedQuery.getResultList();
}
```

Für die zweite Funktion wurde ein CriteriaUpdateBuilder verwendet. Die restliche Implementation blieb fast gleich. Die Limitierung wurde entfernt und ein set() hinzugefügt, um den MQ_IN_STATUS auf NEW zu setzten.

12.1.7 MqInResource

Die Klasse MqInresource beinhaltet den geplanten Endpoint welcher im Arbeitspaket 10.1.4 beschrieben wurde. Die Klasse wurde mit @RestController annotiert, sodass Spring weiss das diese Klasse Endpoints enthält. Sie hat ausserdem noch @RequestMapping, um den Pfad zu definieren und @RequiredArgsConstructor, welcher den Konstruktor generiert mit allen finalen und nicht-null Feldern.

Der Endpoint selbst hat drei Annotationen:

@GetMapping ⁶ deklariert diesen Endpoint als ein GET-Request.

⁶<https://www.geeksforgeeks.org/spring-postmapping-and-getmapping-annotation/>

@Operation ⁷ beschreibt die Funktion von diesem Endpoint.

@RequirePermission ⁸ stellt die Befugnis von diesem Endpoint ein.

Der Endpoint selbst greift nur auf die erstellte Funktion in der Klasse `MqInService.java` zu und mapped das erhaltene Resultat mithilfe des Mappers bevor er es wieder zurück sendet.

Der zweite Endpoint für das ausführen von einem weiteren Verarbeitungsversuch ist gleich aufgebaut wie der erste mit zwei Unterschieden. Er hat zwei andere Annotationen. Einerseits wurde der `@GetMapping` mit einem `@RequestMapping` ersetzt, sodass Spring weiss, dass es die Requests zu den richtigen Endpoints verweisen kann. Ausserdem hat dieser Endpoint ein Parameter namens `id`. Dieser Parameter wird vom `RequestBody` befüllt und muss deshalb mit `@RequestBody` annotiert werden.

12.2 Die Seite und Tabelle im Frontend erstellen

12.2.1 Erstellung der Tabelle

Für die Erstellung der Tabelle wurde eine `mq-in.service.ts`-Klasse erstellt. Diese wird für die Requests an das Backend verwendet und beinhaltet die Funktion `getAllMqInsLimitedTo25()`, um alle `MqIns` von der Datenbank zu laden, und `executeNewProcessingAttempt(id: number)`, die für die erneute Ausführung von einem Verarbeitungsversuch zuständig ist. Beide Funktionen rufen ihren erstellten Endpoint auf, welcher für das Frontend vom Projekt generiert wurde.

Diese Generierung kann mithilfe der Annotationen bei den entsprechenden Endpoints die HTTP-Requests generieren. Durch die generierte Swagger Datei wird eine zweite JSON-Datei generiert, die alle benötigten Informationen beinhaltet und im Modul Interface als `web-backend-admin-api.json` gespeichert wird. Ein Plugin im Frontend nutzt anschliessend diese Datei, um die HTTP-Requests zu generieren.

Die Generierung vereinfachte die Nutzung von HTTP-Requests und hat die entsprechende Implementierung beschleunigt. Die Tabelle für die `MqIns` wurde durch das schnell erstellt und die ersten Einträge waren sichtbar auf der Seite.

⁷<https://www.baeldung.com/swagger-operation-vs-apiresponse>

⁸Vom Projekt CardX implementiert

ERGON - DEV - CardX Web

[Home](#)[Teste starten](#)[Testläufe](#)[Business Daten](#)[Sich logs](#)

admin

Queue-Messages (eingehend)

Datum	Queue-ID	Job	Nachricht	Verarbeitungs-Status	Verarbeitungsergebnis	Anzahl Verarbeitungsversuche	
14.11.2024, 13:21:08	2	2	WorldHelloWorldHelloWorldHello!	ERROR	tschüss	0	---
14.11.2024, 13:19:08	1	1	HelloWorldHelloWorldHelloWorldHelloWorld!	ERROR	Hoi	0	---

Abbildung 12.1: Aktueller Stand der Tabelle

12.2.2 Anzeigen der ganzen Nachricht

Für das Anzeigen der ganzen Nachricht wurde eine neue Map erstellt, welche als Key die ID des MqIns und als Value einen Boolean enthält. Diese Map wird immer beim Aufruf der Seite gefüllt, nach dem die MqIns geladen wurden. Durch diese Map kann die ganze Nachricht angezeigt und anschliessend wieder verkürzt werden. Die eine Funktion setzt den entsprechenden Wert jedes Mal um, wenn man auf den Knopf «ganze Nachricht anzeigen» oder «Nachricht kürzen» drückt. Die kurze Nachricht wird angezeigt:

Job	Nachricht	Verarbeitungs-Status	Verarbeitungsergebnis	Anzahl Verarbeitungsversuche	
2	WH	ERROR	tschüss	0	---
1	HelloWorldHelloWorldHelloWorldHelloWorld!	ERROR	Hoi	0	---

neuer Verarbeitungsversuch starten

ganze Nachricht anzeigen

Abbildung 12.2: Gekürzte Nachricht

Die ganze Nachricht wird angezeigt:

Job	Nachricht	Verarbeitungs-Status	Verarbeitungsergebnis	Anzahl Verarbeitungsversuche	
2	WH	ERROR	tschüss	0	---
1	HelloWorldHelloWorldHelloWorldHelloWorldHelloWorldHelloWorldHelloWorldHelloWorldHelloWorld!	ERROR	Hoi	0	---

neuer Verarbeitungsversuch starten

Nachricht kürzen

Abbildung 12.3: Ganze Nachricht

12.2.3 Erneutes Ausführen vom Verarbeitungsversuch

Die Implementation für dieses Feature dauerte nicht lange, da, wie oben beschrieben 12.2.1, die HTTP-Requests generiert wurden und die Funktion nur noch verknüpft werden musste beim Drücken des Knopfs «neuer Verarbeitungsversuch starten». Dies war schnell gemacht und kostete nur im Backend ein wenig Zeit.

```

private executeNewProcessingAttempt(id: number) {
  this.mqInService.executeNewProcessingAttempt(id)
    .subscribe({
      next: () => {
        this.loadMqIns();
      },
      error: (e: HttpResponse) => {
        throw new ApiHttpResponse('Die Verarbeitung der Queue konnte nicht neu gestartet werden');
      },
    });
}

```

Diese Funktion ruft die Funktion `executeNewProcessingAttempt()` mit dem Parameter ID auf. Anschliessend führt die `Subscribe` folgende zwei Aktionen aus:

next: wird ausgeführt, wenn der HTTP-Request erfolgreich war und die erwarteten Daten bereitgestellt wurden. In diesem Fall wird die Liste neu geladen.

error: wird ausgeführt, wenn der HTTP-Request nicht erfolgreich war und ein Fehler während der Ausführung auftritt. Hier wird ein Fehler mit einer Nachricht geworfen.

Danach ist die Backend-Anfrage fertig und war entweder erfolgreich oder nicht.

12.3 Endpoints für die Erweiterung Filter erstellen

Die Reihenfolge, welche für diesen Abschnitt folgt, ist die Reihenfolge in der implementiert wurde. Die Klassen werden nicht mehr im Detail beschrieben, was oben bereits erklärt wurde.

12.3.1 MqInFilterDto

Um die Datensätze der Datenbank zu filtern, wurde ein weiteres DTO erstellt, um die Filterkriterien vom Frontend zu empfangen. Die Klasse beinhaltet vier Variablen namens `status`, `messageQuery`, `timestampFrom` und `timestampTo`. Ausserdem sind alle Variablen noch mit der Annotation `@Schema` beschrieben.

12.3.2 MqInService

Im Interface `MqInService` wurde wieder eine neue Funktion, `getFilteredMqTablePCs()`, hinzugefügt. Sie nimmt einen Parameter mit dem Typ `MqInFilterQuery`, dieser Record wurde in der Klasse `MqTableDao` hinzugefügt und wird später nochmals genauer erklärt, und gibt eine Liste mit `MqTablePCs` zurück.

12.3.3 MqInServiceImpl

Die Implementierung der neu erstellten Funktion im Interface MqInService wurde hier implementiert. Die Funktion hat gerade mal eine Zeile Code, die im Interface MqTableDao eine weitere Funktion aufruft.

12.3.4 MqTableDao

Das Interface hat zwei neue Inhalte dazu bekommen.

Einerseits wurde die neue Funktion `getFilteredMqTablePCs()` deklariert. Sie hat den gleichen Parameter und Rückgabewert wie die Funktion im Interface MqInService.

Und andererseits wurde hier ein neuer Record namens `MqTableFilterQuery` hinzugefügt. Ein Record ist eine spezielle Art von Klasse, die hauptsächlich für datenzentrierte Klassen dient. Die Klasse ist dadurch einfach und prägnant, da sie Funktionen wie Konstruktoren, Getter, `toString`, `hashCode` oder `equals` bereits implementiert und sie so nicht erneut geschrieben werden müssen. Die Felder der Klasse werden durch Parameter angegeben. Dieser Record beinhaltet `status`, `messageQuery`, `timestampFrom` und `timestampTo` als Parameter.

12.3.5 MqTableHibernateDao

Die zuvor deklarierte Funktion `getFilteredMqTablePCs()` wird hier implementiert. Sie hat ähnliche Variablen wie die anderen Funktionen in dieser Klasse, wie `CriteriaBuilder`, `CriteriaQuery` und `Root`. Was bei dieser Funktion speziell ist, sie hat eine weitere Variable mit dem Typ `Predicate`. Diese Variable ermöglicht es, eine Liste von Filtern zu erstellen. Da nicht immer alle Filter gesetzt werden, müssten ansonsten mehrere Funktionen erstellt werden. Mit dieser Variable kann man die gesetzten Filter zu der Liste hinzufügen und am Ende die Elemente zu der SQL-Query hinzugefügt werden. Dies hat es also ermöglicht, alle Filter mit einer Funktion zu setzen und so das richtige Resultat zu erhalten.

```
@Override
```

```
public List<MqTablePC> getFilteredMqTablePCs(MqTableFilterQuery filterQuery) {  
    CriteriaBuilder cb = getSession().getCriteriaBuilder();  
    CriteriaQuery<MqTablePC> query = cb.createQuery(MqTablePC.class);  
    Root<MqTablePC> mqTable = query.from(MqTablePC.class);  
    List<Predicate> predicates = new ArrayList<>();
```

```
    Optional.ofNullable(filterQuery.status()) // Einfügen der Filters status  
        .ifPresent(status ->  
            predicates.add(cb.equal(mqTable.get(FN_MQ_IN_STATUS), status))  
        );
```

```
    Optional.ofNullable(filterQuery.messageQuery()) // Einfügen der Filters messageQuery  
        .ifPresent(messageQuery -> {  
            String searchTerm = "%" + filterQuery.messageQuery().toLowerCase(Locale.getDefault())  
            predicates.add(  

```



```

        cb.or(
            cb.like(cb.lower(mqTable.get(FN_MESSAGE_SHORT).as(String.class)), searchTerm),
            cb.like(cb.lower(mqTable.get(FN_MESSAGE_LONG).as(String.class)), searchTerm)
        )
    );
});

Optional.ofNullable(filterQuery.timestampFrom()) // Einfügen der Filters timestampFrom
    .ifPresent(timestampFrom ->
        predicates.add(cb.greaterThanOrEqualTo(mqTable.get(FN_MODIFIED_AT), timestampFrom))
    );

Optional.ofNullable(filterQuery.timestampTo()) // Einfügen der Filters timestampTo
    .ifPresent(timestampTo ->
        predicates.add(cb.lessThanOrEqualTo(mqTable.get(FN_MODIFIED_AT), timestampTo))
    );

query.select(mqTable)
    .where(cb.and(predicates.toArray(Predicate[]::new)))
    .orderBy(cb.desc(mqTable.get(FN_MODIFIED_AT)));

return getSession().createQuery(query)
    .setMaxResults(25)
    .getResultList();
}

```

12.4 Die Erweiterung in die Tabelle integrieren

Durch die grosse Verzögerung während dem implementieren von diesem Teil hat sich der Lernende dazu entschieden die Erweiterung zu pausieren und zuerst mit den restlichen Arbeitspaketen weiter zu mache. Falls am Ende der Probe-IPA noch Zeit übrig ist, wird die Implementierung fortgesetzt.

Die Reihenfolge, welche für diesen Abschnitt folgt, ist die Reihenfolge in der Implementiert wurde.

12.4.1 mq-in-filter-url

Um die Parameter von der Url für da Filtern zu extrahieren wurde in dieser Klasse eine Funktion `getMqInFilterUrlStateFromParams` erstellt. Diese Funktion nimmt eine Map mit den Parametern. Mit dieser Map wird mithilfe vom Enum `MqInFilterUrlKeys` alle Werte von der Url hervorgeholt und anschliessend alle Werte, die nichts beinhalten, aus der Liste entfernt. Als Rückgabewert wird `MqInFilterUrlState` verwendet.

Dieser Typ beinhaltet Keys mit dem Typ `MqInFilterUrlKeys` und die Werte haben einen Typ `String`. Dies wird mit einem Record definiert. Durch das hinzufügen des Typs `Partial` werden die Werte optional und müssen nicht befüllt werden.

```

export enum MqInFilterUrlKeys {
  STATUS = 'status',
  MESSAGE_QUEUE = 'messageQuery',
  TIMESTAMP_FROM = 'timestampFrom',
  TIMESTAMP_TO = 'timestampTo',
}

export type MqInFilterUrlState = Partial<Record<MqInFilterUrlKeys, string>>;

export function getMqInFilterUrlStateFromParams(params: ParamMap): MqInFilterUrlState {
  const result: MqInFilterUrlState = Object.values(MqInFilterUrlKeys)
    .reduce((state, urlParam) => ({ ...state, [urlParam]: params.get(urlParam) }), {});
  return omitBy(result, isEmpty);
}

```

Anschliessend wurde noch ein mapping für den Typ `FilterQueryDto` erstellt für die Backend-Anfrage.

12.4.2 mq-in-filter.component

Diese Klasse wird für das anzeigen der Filter verwendet. Es wurde ein `FormGroup` erstellt mit den Filter-Werten, die anfangs einfach Null sind. Der Status jedoch hat den Wert `ERROR` um den Mindestanforderungen gerecht zu werden, da sie bei der ersten Abfrage nur Elemente mit diesem Wert wollen. Dies wird durch ein `ngOnInit()` gemacht, da diese Funktion immer als erstes aufgerufen wird, wenn man die Seite aufruft.

Um die Werte von der Url in das Formular einzufügen wurde eine weitere Funktion erstellt, die alle Parameter mit der Hilfe von der zuvor erstellten Funktion `getMqInFilterUrlStateFromParams()`. Sie speichert die erhaltenen Parameter in einer Variable und setzt anschliessend die `FormGroup` mit den erhaltenen Werten.

Um eine Filterung durchzuführen werden beim Event `submit` alle Werte des Filters im Typ `MqInFilterUrlState` gespeichert und anschliessend für die Elternkomponente bereit gestellt durch ein Event.

Das Endresultat der Implementierten Funktionen und Klassen sieht anschliessend im Web-GUI so aus:

Datum	Queue-ID	Job	Nachricht	Verarbeitungstatus	Verarbeitungsergebnis	Anzahl Verarbeitungsversuche
15.11.2024, 10:17:56	3	3	Nothing!	ERROR		0

Abbildung 12.4: Der aktuelle Stand des Filters

12.4.3 Fehlerbeschreibung

Durch eine Fehler welcher nicht ermittelt werden konnte, wurde die Erweiterung pausiert. Der Fehler erscheint anfangs beim aufrufen der Seite und hat den HTTP-Statuscode 500, was auf einen Server interner Fehler hinweist. Es wurde nicht herausgefunden was der Fehler war. Mit Debugging wurde versucht den Request im Backend abzufangen und manuell durch die Funktionen zu gehen, aber der Request kam nicht im Backend an. Nach ein paar Stunden Debugging und Fehlersuche wurde entschieden, dass die Implementation der Erweiterung pausiert wird und zu einem späteren Zeitpunkt erneut einen Versuch zu starten mit einem Fachverantwortlichen.

12.5 ReleaseNotes schreiben

Um die Änderungen in den Master mergen zu können muss man ReleaseNotes für das implementierte Jira-Ticket schreiben. Für dieses Ticket wurden folgende ReleaseNotes geschrieben:

```
## Webinterface
* CARDXDEV-2268 Neue Seite unter Business Daten > Queue Messages (eingehend)
  hinzugefügt mit einer MqIn-Tabelle.
```

13 Kontrollieren

Dieses Kapitel zeigt die in der IPERKA-Phase «Kontrollieren» durchgeführten Arbeiten auf. In dieser Phase wird beschrieben wie der Lernende die Tests implementiert hat und welche Technologien schlussendlich verwendet wurden.

13.1 Allgemein

Die Tests wurden im entsprechenden Ordner «test» erstellt. In diesem Ordner existieren auch bereits andere Tests und auch Unterkategorien wie configuration, integrationtest, resource, shared und util. Die geplanten Tests wurden in dem Ordner integration und resource erstellt und implementiert. Um die Tests zu schreiben und mit @Test zu deklarieren wurde JUnit ¹ verwendet.

13.2 Integration-Tests

Für die Erstellung der Daten wurde eine interne Klasse von dem CardX-Projekt verwendet. Die Klasse trägt den Namen «A» und ermöglicht es von vielen PCs im Projekt ein Objekt zu erstellen, welches anschliessend in einer Datenbank gespeichert wird und das erstellte Objekt zurückgegeben wird. Diese Klasse wurde verwendet um verschiedene MqTablePC-Objekte zu erstellen und sie daraufhin zu für die Endpoints zu nutzen.

13.2.1 testGetAllMqIns

Dieser Test überprüft die Funktionalität vom GET-Request um alle MqIns mit dem Status ERROR hervorzuholen. Die Objekte werden mithilfe von einer erstellten Funktion erstellt und anschliessend ein GET-Request auf den zu testenden Endpoint gemacht. Dieser Request wird mit WebClient gemacht der von dem Springframework zur Verfügung gestellt wird.

```
MqInDto[] mqInDtos = WebClient.create("http://localhost:" + getPort())
    .get()
    .uri(PATH)
    .retrieve()
    .bodyToMono(MqInDto[].class)
    .block();
```

`.get()` deklariert den Request als GET-Request.

¹<https://junit.org/junit5/>

`.uri(PATH)` wird verwendet um den Pfad des Endpoints anzugeben.

`.retrieve()` wird im zusammenhang mit `bodyToMono()` verwendet.

`.bodyToMono(MqInDto[].class)` stellt sich, dass der Rückgabewert eine Liste von `MqInDtos` ist.

`.block()` blockiert alle synchonen Requests bis dieser Request abgeschlossen ist.

Das Resultat des Requests wurde anschliessend mit «`assertNotNull`» und «`assertThat`» überprüft auf Richtigkeit. Zuerst wurde geschaut ob die erhaltenen Liste null ist und anschliessend die Länge und den Inhalt auf Korrektheit geprüft.

13.2.2 testExecuteNewProcessingAttempt

Das Verfahren der Implementierung dieses Tests sind fast gleich. Der Unterschied ist, dass zweimal ein Get-Request ausgeführt wird. Das erste Mal, um den ursprünglichen Stand der Datenbank hervorzuholen und ein zweites Mal nach dem der PUT-Request getätigt wurde, um zu überprüfen ob das Objekt überschrieben wurde.

Der PUT-Request ist gleich aufgebaut wie die GET-Requests mit zwei kleinen unterschieden. Es wurde ein Body hinzugefügt mit der id des zu überschreibenden Objekts und das `.get()` wurde mit `.put()` ersetzt.

```
Long idOfMqIn = Arrays.stream(mqInsBevorUpdate).findFirst().get().getId();
Integer response = WebClient.create("http://localhost:" + getPort())
    .put()
    .uri(PATH + "/" + idOfMqIn)
    .body(Mono.just(idOfMqIn), Long.class)
    .retrieve()
    .bodyToMono(Integer.class)
    .block();
```

13.2.3 testGetFilteredMqInEntries

Für das Testen der Filter-Requests wurde ein `MqInFilterDto` Objekt erstellt und mit den folgenden Filtern befüllt:

- Nachricht: `mqTable3`
- Status: `NEW`
- Datum bis: `202.11.20TT23:00:00`

Das Filterergebnis sollte alles heraus filtern bis auf ein Element. Dieses wurde anschliessend auf die Richtigkeit des Status, der Nachricht und dem Datum bis.

```
@Test
void testGetFilteredMqInEntries() {
    setupDB();
}
```

```

MqInFilterDto filter = new MqInFilterDto();
filter.setMessageQuery("mqTable3");
filter.setStatus("NEW");
filter.setTimestampTo(LocalDateTime.of(2024, 11, 20, 23, 0, 0).toString());

MqInDto[] mqIns = WebClient.create("http://localhost:" + getPort())
    .put()
    .uri(PATH + "/filtered")
    .body(Mono.just(filter), MqInFilterDto.class)
    .retrieve()
    .bodyToMono(MqInDto[].class)
    .block();

assertNotNull(mqIns);
assertThat(mqIns.length).isEqualTo(1);
assertThat(mqIns[0].getMqInStatus()).isEqualTo(MqInStatus.NEW);
assertThat(mqIns[0].getMessage()).isEqualTo("mqTable3");
}

```

13.3 Unit-Tests

Um die Daten für das Testing bereitzustellen, wurde ein Klasse MqInMock erstellt. Diese Klasse beinhaltet ein paar Funktionen welche ein, mehrere oder verschiedene MqTablePCs zurück gibt.

13.3.1 testMappingWithSinglePC

Dieser Test überprüft die Funktion toMqInDto(MqTablePC mqTablePC) welche aus einem MqTablePC-Objekt ein MqInDto-Objekt mapped. Der Test selbst ist einfach aufgebaut. Er erstellt zuerst das zu mappende Objekt und gibt es anschliessend als Parameter beim Aufruf der Funktion mit. Danach wird mithilfe von assertNotNull() und assertThat() auf überprüft ob es existiert und ob alle Werte richtig gesetzt worden sind.

13.3.2 testMappingWithMultiplePCs

Hier wird ein ähnlicher Ablauf verwendet wie beim Test zuvor. Der Unterschied liegt aber dabei, dass die Funktion listToMqInDto(List<MqTablePc> listOfMqTablePCs) aufgerufen wird und eine Liste mitgegeben wird. Bei der Überprüfung wird ausserdem durch die erhaltene Liste iteriert und bei jedem einzelnen Element die erwarteten Werte überprüft.

```

@Test
void testMappingWithMultiplePCs() {
    // given
    List<MqTablePC> mqTablePCs = MqInMocks.createEveryStatusTypeOfMqTablePC();

    // when

```

```

List<MqInDto> mqInDtos = mapper.listToMqInDto(mqTablePCs);

// then
assertNotNull(mqInDtos);
for (MqTablePC mqTablePC : mqTablePCs) {
    MqInDto mqInDto = mqInDtos.stream().filter(mqIn -> mqIn.getId() == mqTablePC.getId())
    assertNotNull(mqInDto);
    assertEquals(mqInDto.getId(), mqTablePC.getId());
    assertEquals(mqInDto.getMessage(), mqTablePC.getMessage());
    assertEquals(mqInDto.getJobKey(), mqTablePC.getJobKey());
    assertEquals(mqInDto.getModifiedAt(), mqTablePC.getModifiedAt());
    assertEquals(mqInDto.getMqQueueId(), mqTablePC.getMqQueueId());
    assertEquals(mqInDto.getProcessedCnt(), mqTablePC.getProcessedCnt());
    assertEquals(mqInDto.getMqInStatusString(), mqTablePC.getMqInStatusString());
    assertEquals(mqInDto.getMqInStatus(), mqTablePC.getMqInStatus());
}
}

```

13.3.3 testMappingOfMqInFilterDto

Dieser Teste ist gleich aufgebaut wie der erste Test. Die verwendeten Klassen sind andere. Vom größeren Aufbau ist es aber der gleiche Test einfach für eine andere Funktion. Er überprüft die funktionalität der toMqTableFilterQuery Funktion für das Filter der Elemente.

```

@Test
void testMappingOfMqInFilterDto() {
    // given
    MqInFilterDto filterDto = MqInMocks.createMqInFilterDto();

    // when
    MqTableDao.MqTableFilterQuery result = mapper.toMqTableFilterQuery(filterDto);

    // then
    assertNotNull(result);
    assertEquals(result.status(), MqInStatus.ERROR);
    assertEquals(result.messageQuery(), "mqTable1");
    assertEquals(result.timestampFrom(), OffsetDateTime.of(2024, 11, 20, 10, 0, 0, 0));
    assertEquals(result.timestampTo(), OffsetDateTime.of(2024, 11, 20, 20, 0, 0, 0));
}

```

13.4 Codequalität prüfen

Für die Überprüfung von der Codequalität wird das integrierte Jenkins im Projekt CardX verwendet.

Queue-Messages (eingehend)						
Datum	Queue-ID	Job	Nachricht	Verarbeitungs-Status	Verarbeitungsergebnis	Anzahl Verarbeitungsversuche
14.11.2024, 13:21:08	2	2	WorldHelloWorldHelloWorldHello!	ERROR	tschüss	0
14.11.2024, 13:19:08	1	1	HelloWorldHelloWorldHelloWorldHelloWorld!	ERROR	Hoi	0

Abbildung 13.1: Jenkins Codequalität

13.4.1 Falscher Rückgabewert bei setMqInStatusToNew

Die Funktion setMqInStatusToNew() hatte von den PMD-Warnungen aus einen falschen Rückgabewert (int). Diese Warnung tauchte auf wegen dem Namen. Da ein «set» im Namen vorhanden war, wurde vermutet, dass es einen Setter ist, was aber nicht der Fall war. Die Funktion wurde Darauf hin umbenannt zu executeNewProcessingAttempt().

13.4.2 JUnit-Test Klassen müssen final sein

Um die Sichtbarkeit der Testklassen einzuschränken müssen diese Package-private sein. Aus diesem Grund wurde die Sichtbarkeit von MqInIntegrationTest angepasst.

13.4.3 LocalDateTime / -Millis.class

Bei der Erstellung von den Mock-Daten in der Klasse MqInServiceMockImpl wurde der Datums-Typ LocalDateTime verwendet. Die Nutzung von diesem Typ wurde abgeraten und empfohlen den Typ LocalDateTimeMillis zu nehmen, da hier die Millisekunden auch angezeigt werden.

13.4.4 MqInMocks

Die Klasse MqInMocks hat einen privaten Konstruktor und sollte in diesem Fall selbst eine final Klasse sein.

14 Auswerten

Glossar

Backend Der Server-Teil eine Applikation, welcher meistens als Verbindung zwischen der Datenbank und dem Web-GUI genutzt wird. 5

CardX Eine Transaktions-Authorisierungs-Lösung von der Firma Ergon Informatik AG, die bei einigen Banken in der Schweiz im Einsatz ist. 5, 29, 72

Ergon Informatik AG Die Firma, in der ich meine Lehre absolviere. 5, 72

UI Ein UI (User Interface) ist eine Benutzeroberfläche von einer Software oder einem Gerät um mit der Software zu Interagieren. 5

Web-GUI Eine Webseite, die genutzt wird um das Projekt CardX zu überwachen und anzupassen. 5, 72

Abbildungsverzeichnis

1	Logo der Ergon Informatik AG (Ergon Informatik AG 2024)	1
1.1	Anzeigen und bearbeiten der Tasks	7
4.1	Arbeitsplatz des Lernenden	10
5.1	Git Commit History des Quellcodes	13
5.2	Git Commit History der Dokumentation	14
9.1	Ablauf eines GET-Requests für das Erstellen der Tabelle	31
10.1	Klassendiagramm für das Backend	44
10.2	Ablauf eines Filter-Request	48
12.1	Aktueller Stand der Tabelle	60
12.2	Gekürzte Nachricht	60
12.3	Ganze Nachricht	60
12.4	Der aktuelle Stand des Filters	64
13.1	Jenkins Codequalität	70

Quellenverzeichnis

- Bern, ICT Berufsbildung (2024). *Die 6-Schrittmethode IPERKA*. URL: https://www.ict-berufsbildung-bern.ch/resources/Iperka_OdA_200617.pdf (besucht am 06.11.2024).
- Ergon Informatik AG (2024). *Logo der Ergon Informatik AG*. URL: <https://www.ergon.ch/dam/jcr:c42b10d3-a5c7-4ada-b8e5-ccc94f802da3/ergon-logo.png> (besucht am 06.11.2024).