# **Überarbeitung der Angebotserfassung des Transportmanagementsystem HeRo**



|  |  |  |
| --- | --- | --- |
| Auszubildender | Ausbildender Betrieb | Projektbetreuer |
| Lukas Alte-Bornholt | Hellmann Worldwide | Jan Kurpjoweit |
| Rehmstraße 27 | Logistics SE & Co. KG | E-Mail: |
| 49080 Osnabrück  Fachinformatiker Anwendungsentwicklung | Elbestraße 1  49090 Osnabrück | Jan.Kurpjoweit@hellmann.com |
| Prüflingsnummer: 66117 |  |  |

Inhalt

[1. Projektplanung 4](#_Toc84588114)

[1.1. Vorstellung des Unternehmens 4](#_Toc84588115)

[1.2. Werdegang in der Ausbildung 4](#_Toc84588116)

[1.3. Projektbeschreibung 4](#_Toc84588117)

[1.4. Auftraggeber/Motivation 4](#_Toc84588118)

[1.5. Projektschnittstellen 5](#_Toc84588119)

[1.6. Projektabgrenzung 5](#_Toc84588120)

[1.7. Projektphasen 5](#_Toc84588121)

[1.8. Ressourcenplanung 6](#_Toc84588122)

[Entwicklungsprozess 6](#_Toc84588123)

[1.9. Projektziele 7](#_Toc84588124)

[1.10. Datenschutz 7](#_Toc84588125)

[1.11. Ist-Analyse 7](#_Toc84588126)

[1.12. Wirtschaftlichkeit 7](#_Toc84588127)

[1.12.1. Projektkosten 7](#_Toc84588128)

[1.12.2. Amortisation 8](#_Toc84588129)

[1.13. Backlog 8](#_Toc84588130)

[2. Implementierung 8](#_Toc84588131)

[2.1. Beginn des Projekts 8](#_Toc84588132)

[2.2. Umsetzung des Backlogs 8](#_Toc84588133)

[2.2.1. Redesign Angebotsübersicht 9](#_Toc84588134)

[2.2.2. Redesign Angebotsstrecke 10](#_Toc84588135)

[2.3. Qualitätssicherung 10](#_Toc84588136)

[2.3.1. Code-Review 10](#_Toc84588137)

[2.3.2. Testergebnisse 10](#_Toc84588138)

[2.4. Benutzerdokumentation 10](#_Toc84588139)

[3. Fazit 12](#_Toc84588140)

[3.1. Fachliches Fazit 12](#_Toc84588141)

[3.2. Persönliches Fazit 12](#_Toc84588142)

[3.3. Ausblick 12](#_Toc84588143)

[4. Anhang 12](#_Toc84588144)

[4.2. Abbildungen 14](#_Toc84588145)

[4.2.1. Ist-Zustand Angebotserfassung 14](#_Toc84588146)

[4.2.2. Targetdesign Auftragserfassung 14](#_Toc84588147)

[4.2.3. Entwurf der Angebotserfassung 15](#_Toc84588148)

[4.3. Codeabschnitte 16](#_Toc84588149)

[4.3.1. Codeabschnitt 1: HTML und JS der Angebotsübersicht 16](#_Toc84588150)

[4.3.2. Codeabschnitt 2: Bedingte Anzeige der Angebotsnummer 16](#_Toc84588151)

[4.3.3. Codeabschnitt 3: Statusabfrage (getState) und „Beobachter“ (Observer) von Redux 17](#_Toc84588152)

[4.3.4. Codeabschnitt 4: CSS-Klassen des Punkte Abschnitts der Streckenübersicht 18](#_Toc84588153)

[18](#_Toc84588154)

[4.3.5. Codeabschnitt 5: HTML und JS-Code des Punktebereichs der Streckenübersicht 19](#_Toc84588155)

Fettgedruckte Wörter werden im Glossar erklärt.

# Projektplanung

## Vorstellung des Unternehmens

Das Unternehmen Hellmann Worldwide Logistics SE & Co. KG ist ein global agierender Logistikdienstleister. Das Familienunternehmen wurde 1871 von Carl Heinrich Hellmann als Ein-Mann-Betrieb mit Pferdefuhrwerk im norddeutschen Osnabrück gegründet. Derzeit besteht der Vorstand aus Reiner Heiken und Dr. Michael Noth. Der Betrieb ist mit einem weltweiten Netzwerk von 19.500 Beschäftigten in 489 Büros in 173 Ländern vertreten.

## Werdegang in der Ausbildung

Meine Ausbildung zum Fachinformatiker Anwendungsentwicklung habe ich im August 2018 zunächst bei pco GmbH & Co. KG im „Software-Development-Team“ begonnen. Dort wurde mir C++ Desktopentwicklung im Qt-Framework nähergebracht.

Im April 2019 folgte betriebsbedingt ein Betriebswechsel zu Hellmann. Ich habe in dieser Zeit das „Cross Function Team“ , das „AS/400 Team“ und das „Team 429“ sowie „Sheperds‘8“ des „Hellmann Transport Management (HTM) – Teams“ kennengelernt.

Im „Cross-Function Team“ habe ich mithilfe des APEX-Webbaukastens die Abfragesprache SQL beigebracht bekommen. Hauptaufgabe war zur der Zeit die Realisierung einer Meldungs-Software für den Empfang des Gebäudes.

Anschließend im „AS/400 Team“ wurde mir die Großrechnersprache RPG beigebracht, um bei der Entwicklung der Hellas mitzuwirken.

Meine letzte Station war das „HeRo-Team“, welches sich in viele verschiedene Teams aufteilt. Im Team 429 half ich bei der Entwicklung verschiedener Features der Hellscan-App mit. Für mein erstes Abschlussprojekt entwickelte ich im Team „Sheperds‘8“ in JavaScript. Das zweite Abschlussprojekt wurde in dem neu gegründetem Team „UX/DEV“ durchgeführt. Die Programmiersprache ist dieselbe der anderen Teams im HeRo-Team.

## Projektbeschreibung

Das Transportmanagement HeRo ist ein umfangreicher **Webservice** von Hellmann intern entwickelt. HeRo befindet sich in ständiger Wartung und Verbesserung. Verschiedene Bereiche der Applikation sind dabei unterschiedlich weit entwickelt.

Im Bereich der Angebotserfassung besteht zur Zeit am meisten verbesserungsbedarf. Es sollen bestimmte Bereiche der Angebotserfassung in funktionaler und visueller Hinsicht aufbereitet und an gegebene Standards angepasst werden. Die gegebenen Standards kommen aus der Auftragserfassung und sollen in diesem Projekt in die Angebotserfassung migriert werden.

## Auftraggeber/Motivation

Das Projekt war ein Hellmann internes Projekt und wurde vom HeRo-Team in Auftrag gegeben. Es ist deshalb ein Projekt, welches ausschließlich für den Nutzen von Hellmann Worldwide Logistics bestimmt ist.

Im Sommer diesen Jahres fand eine fast vollständige Überarbeitung des User-Interfaces und der User-Experience vom Frontend des HeRo-Transportmanagementsystems statt. Diese Überarbeitung muss nun vollendet werden. Die Angebotserfassung ist ein maßgeblicher Teil des Transportmanagementsystem und muss deshalb zwingend an die neu definierten Standards angepasst werden. Ziel der gesamten Überarbeitung ist eine effizientere und weniger fehleranfällige Nutzung des TMS.

## Projektschnittstellen

Der Projektleiter dieses Projektes ist Jan Kurpjoweit. Er ist der Team Chef und überwacht alle Projekte des Teams. Gleichzeitig übernimmt er die Anforderung und Abnahme dieses Projektes.

Als Ansprechpartner des Projektes sind Jannik Kerzcynski und Mattis Schuermann zuständig. Sie überwachend außerdem laufend den Fortschritt des Projektes und führen die Code-Review sowie Tests durch.

Benutzer von HeRo werden in regelmäßigen Abständen eine neue Version von HeRo erhalten. Deshalb findet regelmäßig ein Austausch über kommende Features statt, um ein konkretes Feedback zu erhalten und gegebenenfalls die Software anzupassen. Dieses Projekt wird, wie Parallelprojekte auch, in Abschnitten ausgerollt. So ist der Benutzer nicht mit allen Features auf einmal konfrontiert und hat Zeit sich einzuarbeiten.

Zu den technischen Schnittstellen gehört Redux und Elasticsearch. Redux ist ein Statuscontainer im Frontend und arbeitet mit Zuständen und Aktionen. Es ist für die lokale Speicherung von Daten zuständig. Elasticsearch ist eine Suchmaschine und zentraler Speicherort für die Nutzdaten von HeRo. Durch Requests werden JSON-Dokumente zwischen Redux und Elasticsearch ausgetauscht. Da Requests und sämtlicher Datenverkehr zwischen den Komponenten mithilfe von JavaScript Code über die REST-Schnittstelle laufen und keine HTTP-Requests benötigen, ist ein erneutes Laden des Webservices nicht nötig, welches die Performance der Webanwendung deutlich erhöht.

## Projektabgrenzung

Die Webapplikation HeRo ist für die weltweite Nutzung und somit alle Features die es beinhaltet, inklusive dieses Projekt. Zur Zeit wird HeRo lediglich an einem Standort von Hellmann Worldwide Logistics aktiv genutzt. Ziel ist es diesen Webservice weltweit auf alle Standorte auszurollen.

Der Projektumfang ist dynamisch aufgebaut, sodass sich der Umfang der zu erledigenden Aufgaben nach der Fortschrittsgeschwindigkeit des Projektes richtet. Dies ist dadurch möglich, dass die gesamten Überarbeitungen der Angebotserfassung nach eigener Einschätzung den maximalen Projektumfang der IHK überschreiten würden.

Man hat sich deshalb dazu entschieden einen leicht umfangreicheren Backlog zu schreiben als die Projektzeit erlaubt und im Projektverlauf zu entscheiden, welche Teilaufgaben noch innerhalb der Projektzeit erledigt werden können. Diese Entscheidung erhöht die Produktivität und Effizienz während der Entwicklung.

## Projektphasen

Der Rahmen des Projektes bietet einen Umfang von 70 Stunden. Die Zeitplanung der einzelnen Projektphasen ist wie folgt geplant.

|  |  |
| --- | --- |
| **Phase** | **Geplante Zeit in Stunden** |
| **Planung und Analyse** | 5 |
| **Anforderungsspezifikation** | 5 |
| **Entwicklung** | 40 |
| **Dokumentation** | 16 |
| **Abnahme** | 4 |
| **Gesamt** | **70** |

Weitere Details sind der detaillierten Zeitplanung im Anhang (\*\*\*\*) zu entnehmen.

* 1. [Ressourcenplanung](https://www.grobmanschwarz.de/leistungen/beratung/)

Für den Zeitraum des Projektes standen 70 Stunden der Arbeitszeit zur Verfügung. Die

Entwicklung fand aus dem Home-Office am Firmen-Notebook statt. Auf dem Notebook ist die

aktuellste Version von Windows 10 Pro installiert.

Entwickelt wurde mit der aktuellen Version von Jetbrains IntelliJ IDEA Ultimate in den Programmiersprachen Javascript und HTML. Im Projekt wird zusätzlich die Basisklasse LitElement zur Erstellung von Webkomponenten verwendet.

Die Versionsverwaltung und das Deploymentverfahren übernimmt der Webservice **Gitlab**.

Als Leitfaden wurden Tickets mithilfe der Jira Webapplikation aufbereitet. Diese dienen als reine Dokumentation für den Projektstatus. Weitere Dokumentationen erfolgen mit Microsoft Word für die Projektdokumentation und Notepad++ für Notizen. Darstellungen von Diagrammen wurden mit Lucidchart durchgeführt.

Zur direkten Kommunikation wird Microsoft Teams verwendet. Absprache und Betreuung erfolgen über Videokonferenzen und Direktnachrichten während des Projektes.

Benötigte Lizenzen für Hard- und Software sind bereits eingekauft und werden nicht zusätzlich berechnet.

* 1. Entwicklungsprozess

Das Projekt wurde nach der **Kanban**-Methode durchgeführt, dabei durchliefen die Tickets nacheinander folgende Stufen:

* Konzeption
* Entwicklung
* Code-Review
* Abnahmetest
* Deployment

In der **Konzeption** wird der Backlog des Projektes vollständig erfasst und in Form von Teilaufgaben niedergeschrieben. Jedes Ticket hat konkrete Anweisungen für das Projekt und bestimmte Abnahmekriterien. In der **Entwicklung** müssen die Tickets nacheinander umgesetzt werden. Nach erfolgreicher Umsetzung findet pro Ticket eine **Code-Review** statt, die den Code auf unterschiedliche Weise überprüft, vergleichbar mit einem White-Box-Test. Um die Code-Review zu entlasten werden in der Entwicklungsphase wichtige Module durch automatische Unittests überprüft. Der **Abnahmetest** überprüft das umgesetzte Ergebnis eines Tickets mit der Ticketbeschreibung. Wenn der gesamte Prozess bis hier erfolgreich war, wird der Code des Tickets deployt. Wenn es Fehler gibt, muss das Ticket die notwendigen Schritte zurückgehen und im schlechtesten Fall neu konzeptioniert werden.

* 1. Projektziele

Sachziel: Teilentwicklung einer visuellen und funktionalen Überarbeitung der Angebotserfassung der TMS-Webapplikation von HeRo.

Zeitziel**:** Für das Projekt sind 40 Arbeitsstunden Entwicklungszeit angesetzt. Die Arbeitszeit von anderen Mitarbeitern wird nicht berücksichtigt.

Qualitätsziel: Die Qualitätssicherung läuft mit der Entwicklung des Projektes in Form von

Code-Reviews und Blackbox-Tests.

## Datenschutz

Die Angebotserfassung zeigt personenbezogene bzw. unternehmensbezogene Daten an. Diese Daten müssen entsprechend der DSGVO verarbeitet und gespeichert werden. Durch mehrere Datenschutzbeauftragte innerhalb des Unternehmens wurden die richtige Verarbeitung und Speicherung der Daten im Vorfeld entwickelt. Der Zugriff von internen Mitarbeitern auf Unternehmensdaten erfolgt durch eine ausführliche Authentifizierung. Durch ein Rechtesystem ist der Lese und Schreibzugriff geregelt.

Um die Angebotserfassung von HeRo zu nutzen ist lediglich der Lesezugriff auf die Daten nötig. Die gesetzlich geforderten Maßnahmen sind vom Unternehmen für dieses Projekt eingehalten.

## Ist-Analyse

Zur Zeit ist die **Angebotserfassung** in seinen Komponenten fertig konzeptioniert, aber weicht vom Target Design und Funktionalität der ähnlich aufgebauten **Auftragserfassung** ab. Dazu gehören die Darstellung Angebotserstellung, die Auflistung der Sendungsdaten sowie der Teilstrecken, die Anzeige und Filterung von weiteren Angeboten und FAB-Buttons (**f**loating-**a**ction-**b**uttons) der Steuerung des gesamten Angebots.

Die aktuelle **Angebotserfassung** bietet nicht genug Informationsdichte. Komplexe Inhalte sind schwierig zu verstehen und können nicht klar genug erschlossen werden. Da eine Applikation wie die Angebotserfassung eine hohe Informationsdichte und einheitliche Struktur erfordert, ist eine entsprechende Aufbereitung zwingend nachzuholen.

## Wirtschaftlichkeit

Die Umsetzung des Projektes wurde aufgrund der geschilderten Anforderungen der Ist-Analyse beschlossen. Im Folgenden wird der wirtschaftliche Aspekt des Projektes betrachtet.

### Projektkosten

Die Projektkosten ergaben sich aus den Personalkosten, den Kosten für die Ressourcennutzung, die Hard- und Software und für benötigte Lizenzen.

Bei der Berechnung der Kosten für die Projektdurchführung werden im Wesentlichen die Personalkosten berücksichtigt. Der Stundensatz basiert auf dem internen Abrechnungssatz, der zwischen der Holdinggesellschaft und der Spartengesellschaft vereinbart wurden.

Hardware und Softwarelizenzen waren bereits vorhanden. Diese Kosten wurden in dieser

Kostenrechnung nicht beachtet. Für die Berechnung der Personalkosten wurde mit einem

Stundensatz von 62,50 € gerechnet. Dies ist der Standardstundensatz für jeden internen Mitarbeiter. Enthalten sind dort bereits die Kosten für die Ressourcennutzung.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Person** | **Tätigkeit** | **Stundensatz** | **Aufwand in Std.** | **Summe** |
| **Lukas Alte-Bornholt** | Projektumsetzung | 62,50€ | 70 | **4375,00 €** |
| **Mattis Schuermann** | Projektunterstützung | 62,50€ | 15 | **937,50 €** |
| **Jannik Kerczynski** | Projektunterstützung | 62,50€ | 15 | **937,50 €** |
| **Jan Kurpjoweit** | Projektbetreuung | 62,50€ | 5 | **312,50 €** |
| **Gesamtsumme:** | | | | **6562,50 €** |

Für das Projekt ergeben sich Gesamtkosten in Höhe von 6562,50€.

### Amortisation

Tagessatz operative speditionelle Mitarbeiter: 350€

## Backlog

Für die Umsetzung des Projektes wurde das Projekt in 5 Abschnitte aufgeteilt. Jeder Abschnitt bildet einen Teilbereich, allerdings nicht jeden, der Angebotserfassung ab. Pro Teilbereich wurde ein Ticket formuliert. Die Überschriften der Tickets lauten:

* Redesign Angebotsübersicht
* Redesign Angebotsstrecke
* Redesign/Workflow Anpassung der floating-action-buttons
* Filter für weitere Angebote funktional umsetzen
* Redesign Weitere Angebote Container

Unter Punkt 2.2 werden die Tickets sowie die Lösung ausführlich beschrieben.

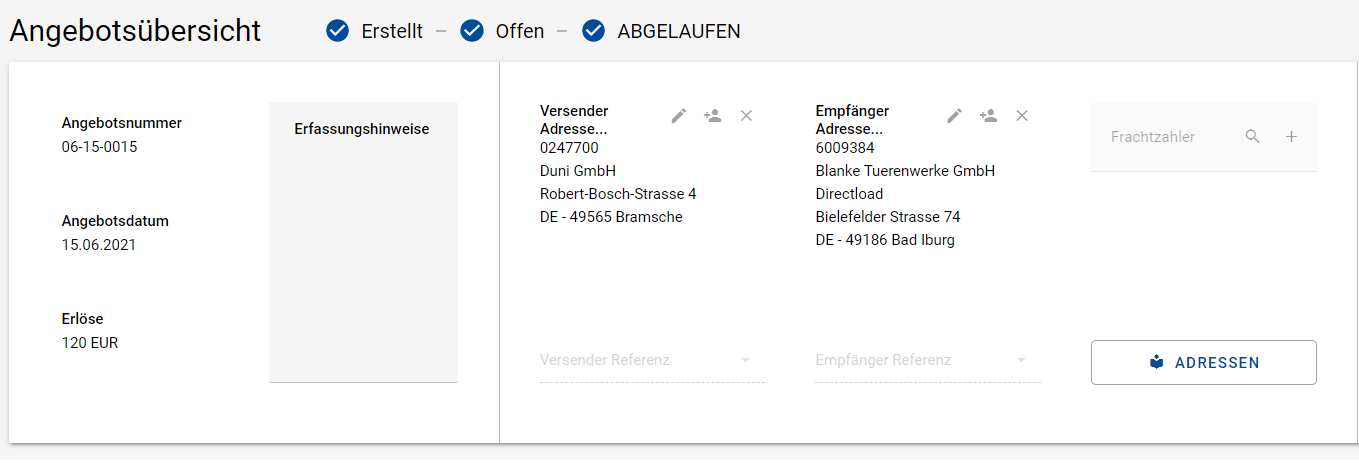
# Implementierung

## Beginn des Projekts

## Umsetzung des Backlogs

Nachfolgend werden alle erledigten Aufgaben beschrieben und mit einer Vorher und Nachher Ansicht veranschaulicht. Vergrößerte Abbildungen sind dem Anhang zu entnehmen.

### Redesign Angebotsübersicht



*Alter Zustand*

In diesem Ticket sollte ein Abschnitt der Angebotsübersicht angepasst werden. Die Felder Disponent, Angebotsdatum und Erlöse sollten untereinander angezeigt werden. Hierbei handelt es sich um reine Anzeigekomponenten, die von Hellmann intern entwickelt wurden. Der Codeabschnitt der rechts abgebildeten Angebotsübersicht befindet sich im Anhang unter Punkt 4.3.1. Durch LitElement kann diese Komponente in eine andere Klassen importiert werden. Der Disponent ist dabei in einem leeren Angebot der aktuelle Benutzer mit Vor- und Nachnamen oder der Ersteller eines bereits gespeicherten Angebots. Das Angebotsdatum ist bei Neuerstellung eines Angebots immer das aktuelle Datum, sonst das Datum zum Zeitpunkt der Erstellung. Das Feld Erlöse zeigt den voraussichtlichen Gewinn des Angebots und ist bei einem neu erstelltem Angebot leer und wird erst bei gespeicherten Angeboten gefüllt.

Außerdem soll in diesem Ticket die Angebotsnummer **nicht** mehr im Feld unten, sondern wie in der Auftragserfassung neben der Überschrift angezeigt werden. Dies ist allerdings nur der Fall, wenn ein Angebot geladen ist. Bedeutet für den html Code, es wird eine konditionale Anzeige in JavaScript gebraucht, die überprüft, ob das Feld Angebotsnummer gefüllt ist. Ein Codeabschnitt ist im Anhang unter Punkt 4.3.1 zu finden. Der ternäre Operator von JavaScript verkürzt das If-Statement.

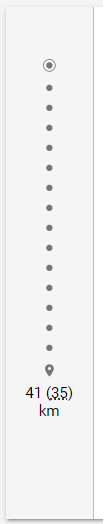
Wenn ein bereits erstelltes Angebot angezeigt werden soll, können die erforderlichen Daten mit Redux geladen werden. Redux ist ein Zustandsspeicher, der mit Aktionen und Zuständen seine Daten verwaltet. Dabei werden Aktionen und Zustände immer zusammen gespeichert, sodass eine nachvollziehbare Zustandskette entsteht. Seine Daten bekommt Redux aus Elasticsearch wie in der Projektplanung unter Schnittstellen beschrieben.

Wenn ein Angebot nun geladen werden soll, wird im Code auf eine Action von Redux gehorcht. Diese Action liefert einen neuen Status und damit die entsprechenden Daten, die dann in die Variablen im HTML-Code übergeben werden. Ein Codeabschnitt dazu ist im Anhang unter Punkt 4.3.2 zu finden. Der Lifecycle von LitElement sieht vor die „connected callback“ vor dem „constructor“ auszuführen. Der Observer wird im Code auf eine bestimmte Variable gesetzt (bind). Anschließend fragt der „constructor“ die Variable ab und erhält seinen State. In den States stehen dann die Daten.



*Neuer Zustand*

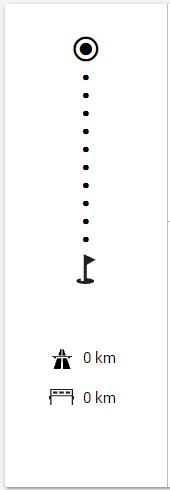
## Redesign Angebotsstrecke



*Alter Zustand*

Das Redesign der Angebotsstrecke beinhaltet die Erneuerung des Containers für die Streckenübersicht der Angebotserfassung. Jan Kurpjoweit hat im September einen Entwurf für die Angebotserfassung gefertigt, sodass mit der Umsetzung dieses Entwurfs zu Teilen begonnen werden konnte. Erneuert werden soll nun die Kilometeranzeige sowie die Anpassung der Größe des gesamten Containers, dazu soll eine Überschrift mit Icon. Die gesamte Streckenübersicht im alten Zustand ist der Abbildung unter Punkt 4.2.1 im Anhang zu entnehmen. Der gesamte Entwurf ist zur Orientierung im Anhang unter Punkt 4.2.3 zu finden. Die Zahl unter den Punkten bedeutet tatsächliche Kilometer, die Zahl in Klammern Mautkilometer.

Insgesamt muss dieser Bereich breiter gemacht werden. Außerdem ist es nicht eindeutig welche Zahl für welche Art von Kilometern steht. Es ist logisch, dass die kleinere Zahl immer die Mautkilometer sein müssen, dennoch soll es dem Benutzer einfacher gemacht werden, sie zu unterscheiden. Deshalb wurden zwei Icons ausgewählt, die die Bedeutung der Kilometer abbilden. Das Icon für die tatsächlichen Kilometer wird außerdem für das Icon in der Überschrift verwendet. Die Punkte über den Kilometern werden insgesamt größer gemacht und sind in schwarz für mehr Kontrast. Weiter hervorgehoben wird diese Abbildung durch einen weißen Hintergrund, dadurch gehört dieser Teil mehr zum gesamten Inhalt der Angebotsstrecke. Das Icon oberhalb der Punkte bleibt bestehen, unterhalb der Punkte wird das Icon durch eine Fahne ausgetauscht.



*Neuer Zustand*

Die beschriebene Komponente ist eine alleinstehende Klasse und wird in die Oberklasse eingefügt. Die Daten kommen wie in der Angebotsübersicht von Redux. Es werden auf Zustandsänderungen gehorcht und anschließend die Daten übertragen und in die Variablen gefüllt. Ein Codeabschnitt vom „constructor“ und der „connectedCallback“ wird nicht eingefügt, da das Prinzip in der Angebotsübersicht erklärt wird.

Für diese Komponente mussten hauptsächlich CSS-Klassen angepasst werden, ein Codeabschnitt von den CSS Klassen ist deshalb im Anhang unter Punkt 4.3.4 zu finden. Ein einheitliches Design wird durch die Variablen deutlich vereinfacht (zum Beispiel: *var(--hellmann-margin-small)*). Der HTML-Code ist im Anhang unter Punkt 4.3.5 zu finden.

## Qualitätssicherung

Die Kanban Methode sieht vor für jedes visualisierte Feature eine separate Qualitätssicherung vorzunehmen, um möglichst genau eine mögliche Fehlerquelle abzudecken. Deshalb werden pro Feature vom Entwickler eine Code-Review und von einem zweiten, unabhängigen Entwickler die Blackbox-Tests durchgeführt.

## Funktionstests während der Entwicklung

Für jedes Ticket wurden eine Reihe von Meilensteinen und White-Box-Tests durchgeführt, die im Folgenden aufgelistet werden.

* + - 1. Funktionstest Redesign Angebotsübersicht

|  |  |  |
| --- | --- | --- |
| Bereich | Beschreibung | Ergebnis |
|  |  |  |
|  |  |  |
|  |  |  |

* + - 1. Funktionstest Redesign Angebotsstrecke

|  |  |  |
| --- | --- | --- |
| Bereich | Beschreibung | Ergebnis |
|  |  |  |
|  |  |  |
|  |  |  |

## Code-Review

Die Code-Review beginnt mit der Erstellung eines **Merge-Requests**. Der Merge-Request beinhaltet immer einen Source-Branch und einen Ziel-Branch. Jedes Mal wenn Commits auf dem Source-Branch hochgeladen werden, fängt **Gitlab** an den gesamten Code in einer **Merge-Request-Pipeline** zu überprüfen. Dabei werden insgesamt 6 Stages durchlaufen:

Für Transparenz und Fehlersuche hat jede Stage ein eigenes Log. Die Überwachung des Pipeline Prozesses ist fester Bestandteil und Hauptwerkzeug der Code-Review. Ein Entwickler untersucht den Code bei einer Code-Review nach folgenden Kriterien:

* Code-Qualität
* Vergleich der Anforderungen
* Namenskonventionen
* Fachliche Konventionen
* Abhängigkeiten
* Technische und fachliche Umsetzung
* Testabdeckung (Unittests)

Wenn eine **Pipeline** erfolgreich durchlaufen ist, wird der Merge-Request zum Mergen freigegeben unter der Voraussetzung, dass ein zweiter, berechtigter Entwickler diesen Merge-Request genehmigt hat.

Um die entwickelten Features zu testen, muss der Code in die Testumgebung verschoben werden. Die Testumgebung ist dazu da, um mehrere Features in Zusammenarbeit zu testen. Um auf die Testumgebung zu deployen muss eine Feature-Umgebung erstellt werden. Hier werden einzelne Features getestet. Auf dieser Umgebung laufen End2End-Tests, bei Erfolg ist der Deployment-Prozess genehmigt und der Code kann in die Testumgebung integriert werden.

## Benutzerdokumentation

# Fazit

## Fachliches Fazit

## Persönliches Fazit

## Ausblick

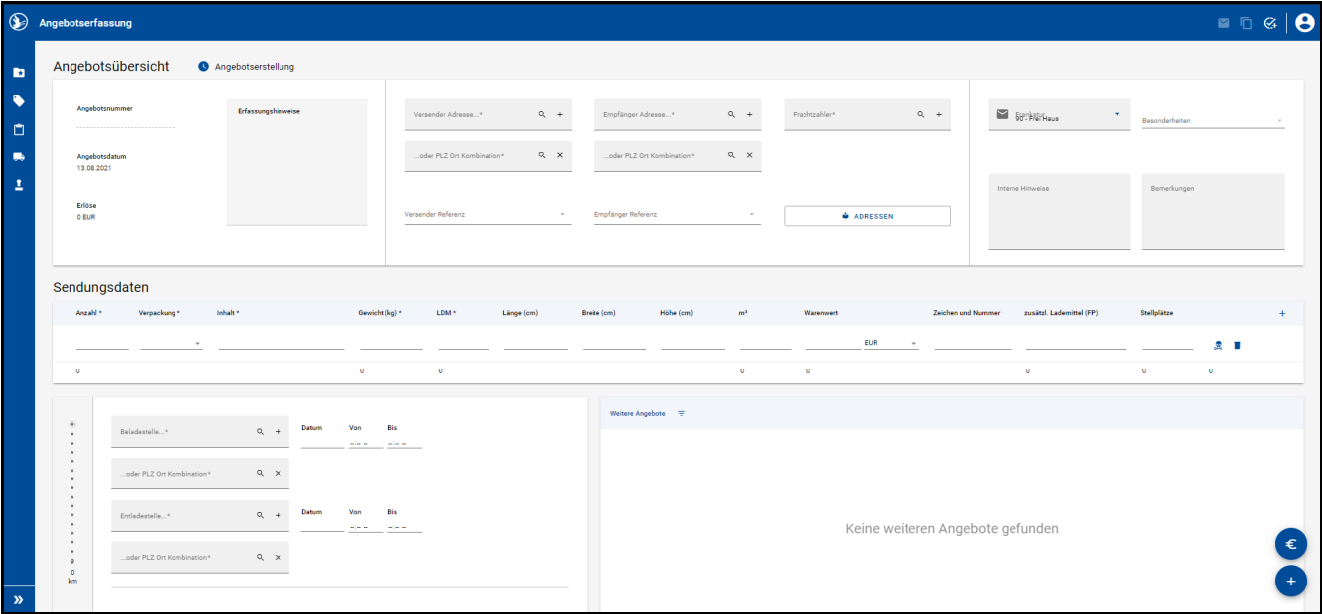
# Anhang

#### Glossar

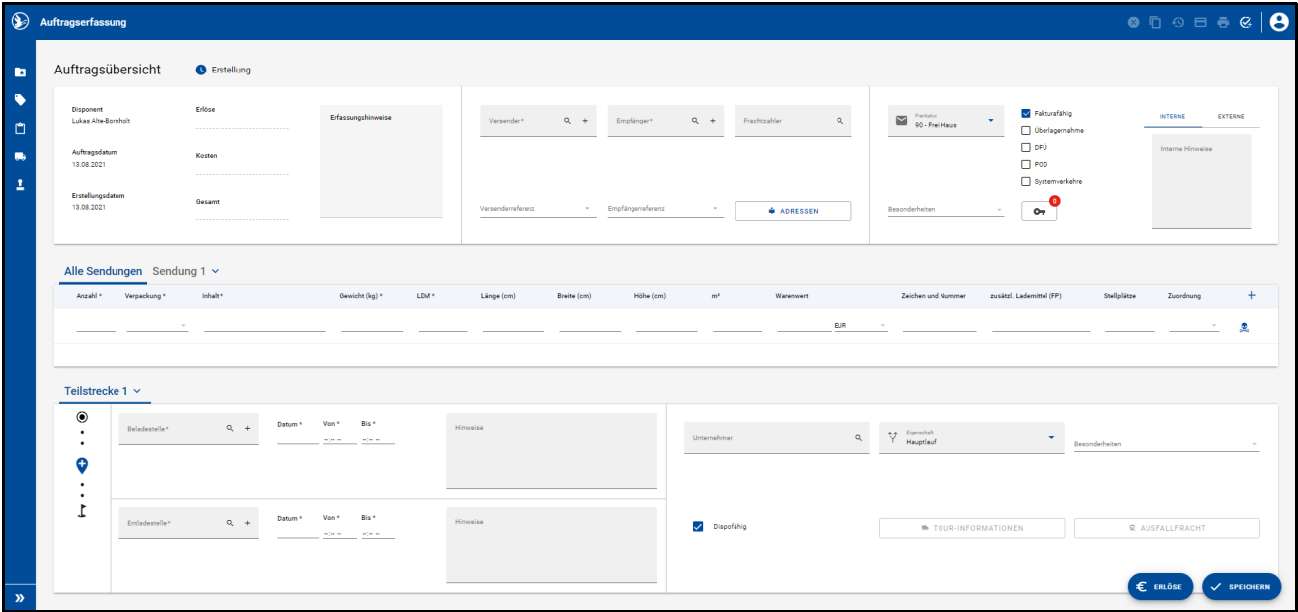
|  |  |
| --- | --- |
| **Begriff** | **Bedeutung** |
| **Merge-Request** | Anfrage zur Einarbeitung eines Sourcebranches in einen Targetbranch. |
| **Merge-Request-Pipeline** | Eine spezielle Art von Pipeline extra für Merge-Requests. Es handelt sich dabei um eine detached-Pipeline, welche sich ausschließlich den Code für den Merge-Request (Source- mit Targetbranch) überprüft. |
| **Pipeline** | Eine Aneinanderreihung von Stages, die immer in der festgelegten Reihenfolge ihre jeweiligen Jobs ausführen(z.B. Bauen der Anwendung). Eine Stage kann mehrere Jobs enthalten. |
| **Instanz, Umgebung** | Eine Produktumgebung und Teil des Deployment-Prozesses von Gitlab. HeRo verwendet die Instanzen Test, UAT (Staging) und Prod (Live-Umgebung). Zusätzlich existieren unterschiedlich viele Feature-Umgebungen je nach Bedarf. |
| **Gitlab** | Komplexe Webanwendung für Development-Prozesse. Hauptbestandteil ist ein Repository-Manager basierend auf Git, welcher die Funktionen Continuous Integration, Issue Tracking und Deployment-Pipeline bereitstellt. |
|  |  |
|  |  |

## Abbildungen

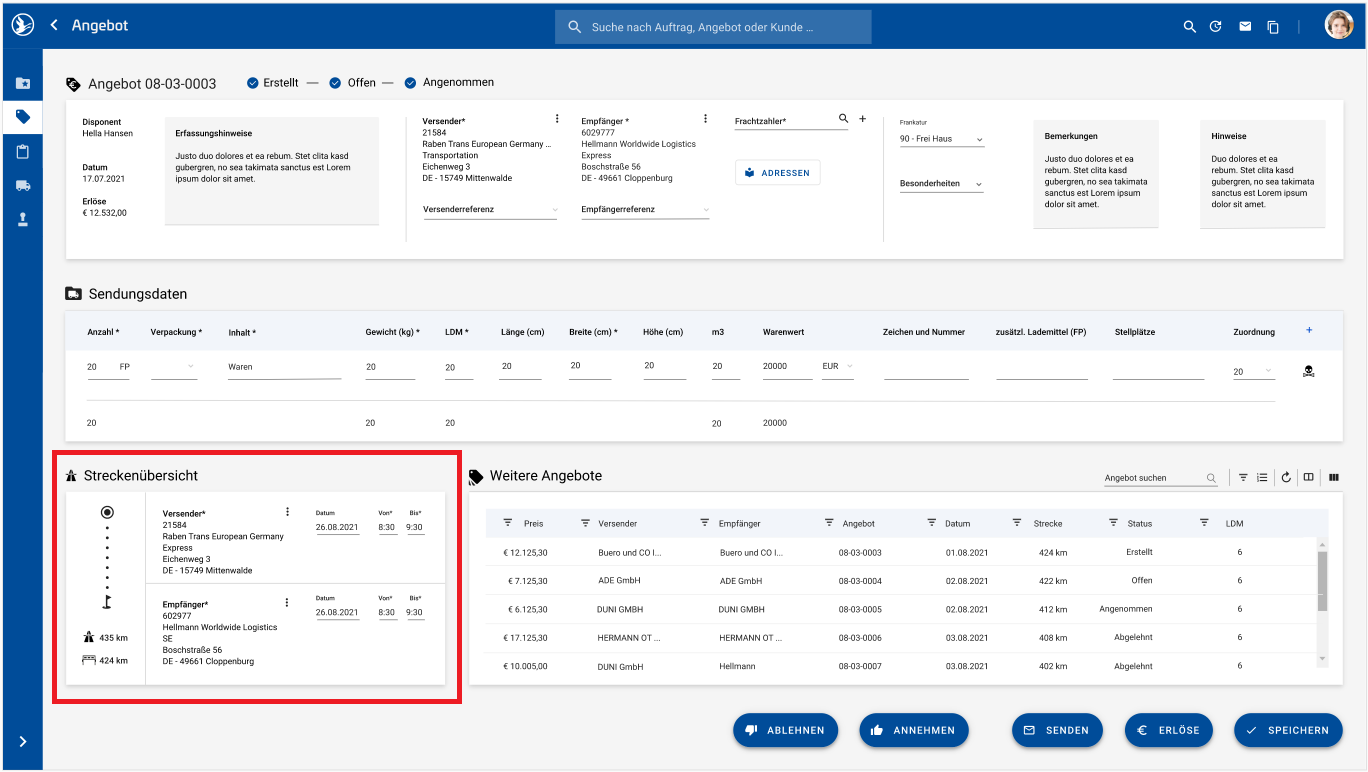
### Ist-Zustand Angebotserfassung



### Targetdesign Auftragserfassung



### Entwurf der Angebotserfassung



## Diagramme

## Detaillierte Zeitplanung

|  |  |  |
| --- | --- | --- |
| **Phase** | **Aufgabe** | **Aufwand in Stunden** |
| **Planung und Analyse** | Ermittlung des aktuellen Entwicklungsstandes | 3 |
|  | Anforderungsanalyse | 2 |
|  | **Zwischensumme** | **5** |
| **Anforderungsspezifikation** | Verfassung Backlog | 3 |
|  | Aufwandsschätzung | 1 |
|  | Erstellung eines Projektplans | 1 |
|  | **Zwischensumme** | **5** |
| **Implementierung** | Redesign Angebotsübersicht | 6 |
|  | Redesign Angebotsstrecke | 5 |
|  | Redesign/Workflow Anpassung FAB-Buttons | 8 |
|  | Filter weitere Angebote | 9 |
|  | Redesign Container weitere Angebote | 7 |
|  | **Zwischensumme** | **35** |
| **Qualitätssicherung** | Code-Review von Merge-Requests | 1 |
|  | Definition Unittests | 3 |
|  | Durchführung der Tests von einem weiteren Entwickler | 1 |
|  | **Zwischensumme** | **5** |
| **Dokumentation** | Kundendokumentation | 2 |
|  | Projektdokumentation | 14 |
|  | **Zwischensumme** | **16** |
| **Abnahme** | Vorstellung der Features an vorgesetzte Mitarbeiter | 2 |
|  | Rollout der Features durch Deploymentinstanzen | 2 |
|  | **Zwischensumme** | **4** |
| **Gesamt** | | **70** |

## Codeabschnitte

### Codeabschnitt 1: Layout und Variablen der Angebotsübersicht

render() {  
 return ***html***`  
 <div class="kurzinfo-grid">  
 <hwl-ui-data-display  
 id="disponent"  
 class="disponent"  
 .label="${this.localize('Disponent')}"  
 .value="${this.\_showErsteller()}"  
 >  
 </hwl-ui-data-display>  
 <hwl-ui-data-display  
 id="angebotsdatum"  
 class="angebotsdatum"  
 label="${this.localize('Angebotsdatum')}"  
 value="${this.\_isAngebotsdatumEmpty(this.angebotsdatum) ?  
 this.\_formatDate(new ***Date***()) : this.\_formatDate(this.angebotsdatum)}"  
 >  
 </hwl-ui-data-display>  
 <hwl-ui-data-display  
 id="erloese"  
 class="erloese"  
 .label="${this.localize('Erloese')}"  
 .value="${Type.*isPresent*(this.erloese) ? this.\_showErloese() : 0}">  
 </hwl-ui-data-display>  
 <div class="erfassungshinweisContainer erfassungshinweise">  
 <p class="erfassungshinweisLabel">${this.localize('erfassungshinweise')}</p>  
 <div class="erfassungshinweisValue-container">  
 <p class="erfassungshinweisValue">${this.erfassungshinweise}</p>  
 </div>  
 </div>  
 </div>  
 `;  
}

### Codeabschnitt 2: Bedingte Anzeige der Angebotsnummer

<div class="angebotsübersicht">  
 <div class="headerContainer">  
 ${

this.\_isAngebotsnummerEmpty() ?  
 ***html***`<div class="title">${this.localize('angebotsuebersicht')}</div>` :   
 ***html*** ` <span class="title">${this.localize('Angebot ')

+ this.angebotsnummer}</span>`

}

### Codeabschnitt 3: Statusabfrage (getState) und „Beobachter“ (Observer) von Redux

constructor() {  
 super();  
 this.loadResources(***Locales***);  
  
 this.statusTexte = {};  
  
 this.\_onAngebotsDatumChanged(this.getState('angebot.data.angebot.angebotsdatum'));  
 this.\_onErfassungsHinweiseChanged(this.getState('angebot.data.angebot.erfassungshinweise'));  
 this.\_onErloeseChanged(this.getState('angebot.data.angebot.erloese'));  
 this.\_onKundeIdChanged(this.getState('angebot.data.erfassungsHinweiseKundeId'));  
 this.\_onKundeChanged(this.getState('angebot.ui.kurzinfo.erfassungsHinweiseKunde'));  
}  
  
connectedCallback() {  
 super.connectedCallback();  
  
 this.setObserver({  
 'angebot.data.angebot.angebotsdatum': this.\_onAngebotsDatumChanged.bind(this),  
 'angebot.data.angebot.erfassungshinweise': this.\_onErfassungsHinweiseChanged.bind(this),  
 'angebot.data.angebot.erloese': this.\_onErloeseChanged.bind(this),  
 'angebot.data.erfassungsHinweiseKundeId':this.\_onKundeIdChanged.bind(this),  
 'angebot.ui.kurzinfo.erfassungsHinweiseKunde': this.\_onKundeChanged.bind(this),  
 });  
}

### Codeabschnitt 4: CSS-Klassen des Punkte Abschnitts der Streckenübersicht

### 

static get *styles*() {  
 return ***css***`  
 .strecke-container {  
 display: grid;  
 justify-items: center;  
 grid-row-gap: 36px;  
 padding: 24px;  
 }  
   
 .strecke-icons {  
 display: grid;  
 justify-items: center;  
 row-gap: 8px;   
 margin-bottom: var(--hellmann-margin-small);  
 }  
  
 .strecke-text {  
 text-align: center;  
 font-family: var(--hellmann-font-family);  
 font-size: var(--hellmann-font-size);  
 }  
   
 .circle {  
 color: var(--hellmann-text);  
 font-size: 7px;  
 }  
  
 .end svg {  
 height: 24px;  
 width: 24px;  
 }  
   
 .kilometer-container {  
 display: flex;  
 flex-direction: column;  
 gap: 16px;  
 }  
   
 .strecke-grid {  
 font-family: var(--hellmann-font-family);  
 font-size: var(--hellmann-font-size);  
 font-weight: var(--hellmann-font-weight);  
 display: grid;  
 grid-template-columns: 10px auto;  
 gird-template-rows: 1fr;  
 column-gap: 12px;  
 justify-items: center;  
 }  
 `;  
}

### Codeabschnitt 5: HTML und JS-Code des Punktebereichs der Streckenübersicht

render() {  
 return ***html***`  
 <div class="strecke-container">  
 <div class="strecke-icons">  
 <mwc-icon id="start\_icon" class="start">radio\_button\_checked</mwc-icon>  
 ${this.\_getPunkteArray(this.punkte).map(  
 item => ***html***` <mwc-icon class="circle">fiber\_manual\_record</mwc-icon> `  
 )}  
 <mwc-icon id="end\_icon" class="end" >${***ziel***}</mwc-icon>  
 </div>  
  
 <div class="kilometer-container">  
 <div id="strecke" class="strecke-grid">  
 ${***autobahn***}   
 ${ this.tatsaechlicheKilometer}   
 ${this.localize('km')}  
 </div>  
 <div id="strecke-maut" class="strecke-grid">  
 ${***maut***}  
 ${this.tatsaechlicheMautkilometer}   
 ${this.localize('km')}  
 </div>  
 </div>  
 </div>  
 `;  
}